Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Desenvolupament d'un videojoc estil JRPG tàctic

Document: Memòria

Alumne: Arnau Marquez Suñer

Tutor: Gustavo Patow
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any): Juny 2021

Índex

1. Introducció **5**

1.1 Motivacions *6*

1.2 Propòsit *6*

1.3 Objectius *6*

1.4 Distribució de tasques *6*

2. Estudi de viabilitat **8**

2.1 Públic objectiu *8*

2.2 Fonts de referència *8*

2.3 Estat de l'art *9*
2.3.1 Cerques 9
2.3.2 Paràmetres d'avaluació 12

2.4 Viabilitat del projecte *13*
2.4.1 Viabilitat tecnològica 13
2.4.2 Viabilitat econòmica 13
2.4.3 Viabilitat legal 14
2.4.4 Conclusions 14

3. Planificació **15**

3.1 Paquets de treball *15*
3.1.1 Documentació 15
3.1.2 Mecàniques de combats 15
3.1.3 Mecàniques del món 15
3.1.4 Mons del joc 15
3.1.5 Interfícies d'usuari sense impacte en la jugabilitat 16
3.1.6 Interfícies d'usuari dins el món del joc 16
3.1.7 Art del joc 16
3.1.8 Test 16

3.2 Cronograma *16*

3.3 Metodologia i Seguiment *17*

4. Marc de treball i conceptes previs **18**

4.1 Referències i conceptes *18*
4.1.1 Gènere JRPG 18
4.1.2 Món del joc 20
4.1.3 Missions i Equipament 20
4.1.4 Escenari del joc 20

4.2 Entorns de treball *21*

5. Disseny del videojoc **22**

5.1 Narrativa *22*

5.2 Personatges *22*

5.3 Espai i escenaris *25*

5.4 Mecàniques *27*
5.4.1 Objectes 27
5.4.2 Accions 28
5.4.3 Reptes 30

5.5 Economia del joc *33*
5.5.1 Recursos 33
5.5.2 Entitats 34
5.5.3 Funcions 35
5.5.4 Representació de les interfícies 36

5.6 Interfícies *37*
5.6.1 Menú inicial 37
5.6.2 Menú de pausa 38
5.6.3 Menú de missions i botiga 38
5.6.4 Interfície del món 39
5.6.5 Interfície de combat 40
5.6.6 Menú final 43
5.6.7 Principis de Mayer 43

5.7 Ambients *45*

5.8 Elements a desenvolupar *47*

5.9 Adaptabilitat *48*

5.10 Flowchart *48*

6. Implementació i proves **49**

6.1 Organització *49*

6.2 Objectes *50*

6.3 Funcions de l'economia del joc *52*

6.4 Entorn *56*

6.5 Combats *58*
6.5.1 Moviment 58
6.5.2 Habilitats 59

6.6 Personatges *68*
6.6.1 Controls Combats 68
6.6.2 Animacions 70
6.6.3 Grups d'enemics 71

6.7 Càmera *72*

6.8 Missions *72*

6.9 Botiga i equipament *73*

7. Resultats **74**

7.1 Creació d'un mapa *74*

7.2 Desplaçament pel mapa *74*

7.3 Sistema d'experiència i nivells *75*

7.4 Creació de grups d'enemics aleatoris *76*

7.5 Combats en el món del joc *76*

7.6 Moviments dels combats *77*

7.7 Habilitats dels combats *77*

7.8 Varietat d'enemics i caps *80*

7.9 Menús *80*

7.10 Interfícies d'usuari *81*

7.11 Interfícies i animacions de món i de combat *82*

7.12 Altres *82*

7.13 Errors durant el desenvolupament *82*

8. Conclusions **84**

9. Treball futur **85**

10. Bibliografia **86**

11. Annexos **88**

11.1 Funció discretització del camí del jugador *88*

11.2 Funcions moviments combat personatge *93*

11.3 Funcions rang atac personatges *95*

11.4 Intel·ligència artificial enemics *104*

12. Manual d'usuari i d'instal·lació **106**

12.1 Manual d'instal·lació *106*

12.2 Manual d'usuari *106*

1. *Introducció*

El projecte desenvolupat consisteix en la creació d'un videojoc estil JRPG tàctic, molt freqüents en el mercat asiàtic, com el seu nom indica; Japanese Role-Playing Game, malauradament, hi ha pocs jocs d'aquest gènere desenvolupats a Europa.

El nom del videojoc és Kingdom Doom. Està desenvolupat en 3D sobre el motor de videojocs Unity, com veurem més endavant. El joc té uns referents molt clars i en cap moment del desenvolupament s'allunya d'aquests.

Figura 1: Captura del Kingdom Doom, el joc desenvolupat per el TFG

Aquest videojoc ve molt influenciat per un videojoc francès molt conegut, anomenat Dofus. Com veurem al llarg de la memòria, el sistema de combats és molt similar.

Figura 2: Captura del Dofus

1.1 *Motivacions*

Quan algú s'endinsa en el món de la programació, ja no veu els programes o videojocs com els veu un usuari normal, es comença a fer preguntes del funcionament del programa, busca errors i solucions a aquests, o quan alguna cosa és interessant, busca una manera de com seria possible de dur a terme, etc. Exactament per aquest motiu va sorgir la idea de crear el joc, arran de jugar molt i gaudir d'un joc. En aquest cas, les preguntes sorgien soles i el TFG era una bona oportunitat per descobrir com es podia crear un joc d'aquestes característiques.

1.2 *Propòsit*

Tenint en compte que al marcat europeu hi ha poca competència en els jocs de l'estil JRPG, és un bon gènere per explorar. Tot i que el propòsit principal és aprendre sobre la creació de videojocs i agafar experiència en el desenvolupament, tots els coneixements adquirits sempre poden ser útils per projectes futurs.

1.3 *Objectius*

L'objectiu d'aquest Treball Final de Grau, és desenvolupar un prototip d'un videojoc del gènere JRPG, que permeti una experiència propera al joc final en tot el que s'hagi pogut desenvolupar en el temps disposat. Un altre objectiu és ser fidels al gènere esmentat i que el joc contingui el màxim d'aspectes relacionats.
El joc serà desenvolupat amb el motor gràfic Unity, així com el GIMP per a l'edició d'icones i imatges.

1.4 *Distribució de tasques*

Per al desenvolupament d'aquest videojoc, l'equip consta d'una única persona que s'haurà d'encarregar de les diferents parts. A continuació es mostra un quadre (Taula 1) amb una valoració personal sobre el valor i l'esforç dedicat als quatre elements principals del joc.

Estètica	25%
Narrativa	5%
Mecàniques	30%
Tecnologia	40%

Taula 1 Quadre d'autoavaluació

L'element en el qual es dedicaran més recursos és la tecnologia. Aquesta etapa busca un correcte desenvolupament a partir del motor gràfic Unity amb les llibreries que calguin per a una millora en el funcionament del joc.

El segon element més importat és el de mecàniques, ja que el correcte funcionament dels combats i del desplaçament del món i la complexitat d'aquestes dues coses requereix una gran atenció.

L'estètica del joc també requerirà certa dedicació, ja que un joc sempre necessita una bona carta de presentació i el que primer es veu d'un joc és l'estètica, tant dels menús com del món.

Per últim tenim la narrativa, ja que amb el temps que disposem no serà molt elaborada, donant una gran importància a les mecàniques i fent que el jugador gaudeixi del joc, i no de la història.

2. *Estudi de viabilitat*

En aquest apartat es valorarà la viabilitat del projecte segons els recursos necessaris, els requisits tecnològics i el cos econòmic.

2.1 *Públic objectiu*

El perfil principal dels jugadors als quals està orientat el videojoc és per gent que juga habitualment a jocs de rol, com també a gent que vol descobrir el gènere. El públic objectiu no és molt ampli, ja que el gènere JRPG és molt concret i els jocs de rol necessiten una certa dedicació per aconseguir tot el que el joc demana per ser completat.

Tot i això, el joc està desenvolupat perquè els combats no siguin extremadament llargs, així que permet ampliar una mica més el públic objectiu, tot i que per completar el joc es necessita temps per a millorar el personatge i les seves habilitats.

Els jugadors no requeriran cap coneixement previ, ja que tot el que han de conèixer serà donat dins el joc. Hi haurà certes coses que caldrà descobrir a mesura que es va avançant de nivells i en la història del joc, requerirà la interpretació del jugador per aprendre'n el funcionament.

2.2 *Fonts de referència*

Les cerques més interessants per desenvolupar el joc, seria qualsevol que estigui relacionada amb els següents aspectes del joc:

- Joc 3D
- Vista isomètrica
- Gènere JRPG o RPG (Joc per torns)
- Joc estètica cartoon

Utilitzant les paraules anteriors, tant soles com combinades entre si, és la forma en què s'han dut a terme les cerques. D'aquesta manera, podíem recollir informació tant dels aspectes que han de tenir les mecàniques, com l'estètica que volem que tingui el joc.

2.3 *Estat de l'art*

2.3.1 *Cerques*

Aquest és un recull de les cerques que s'han fet per recollir informació sobre la creació del videojoc, tant de la part de les mecàniques com de l'estètica del videojoc.

- Cerca 1:
    - Paraules Clau: What makes a rpg game
    - On: Google
    - Resultats: 240.000.000
    - Representatius:
        - What is a Role-Playing Game (RPG)? – Definition [1]
        - What Are Role-Playing Games Even? How are they that? [2]

- Cerca 2:
    - Paraules Clau: Turn based rpg
    - On: Google
    - Resultats: 76.700.000
    - Representatius:
        - 15 Of The Best Turn-Based RPG's Of All Time [3]
        - 12 recent games that prove turn-based RPGs aren't dead [4]

A més a més de les cerques, s'han contemplat jocs de referència que ja teníem presents i han estat molt útils per l'orientació del desenvolupament del joc. Els jocs més influents han estat els següents:

- **Dofus:** És un videojoc en 2.5D, del gènere JRPG tàctic, amb un sistema de lluites molt elaborat. Dofus és el joc que més ha inspirat aquest projecte.

Figura 3: Captura del Dofus

- **Wakfu:** Aquest videojoc està creat per la mateixa empresa que Dofus. Podríem dir que és un germà d'aquest, ja que el sistema de combats és pràcticament idèntic, però és gratuït i té molts canvis en el funcionament del món.

Figura 4: Captura del Wakfu

- **Final Fantasy Tactics:** En aquest cas, és un joc 2D, offline i lineal, no com els dos anteriors. Està molt basat en els combats, els quals són molt similars als dos jocs anteriors, però no tant elaborats.

Figura 5: Captura del Final Fantasy Tactics

- **Dragon Quest Tact:** És el videojoc més recent, per a dispositius mòbils i en 3D. Aquest videojoc és el que més s'acosta tant per les mecàniques de combat com per l'art del joc.

Figura 6: Captura del Dragon Quest Tact

- **Dragon Quest VIII:** És un joc JRPG típic, és a dir, el sistema de combats és molt més simple, ja que només permet atacar o prendre una sola acció cada torn, però l'estètica del joc és similar al resultat que busquem.

Figura 7: Captura del Dragon Quest VIII

2.3.2 *Paràmetres d'avaluació*

Compararem els jocs esmentats en l'apartat anterior amb el joc desenvolupat, els aspectes a comparar seran els següents:

- Originalitat
  - 0: No innova en cap dels seus aspectes.
  - 10: Una idea nova, que innova en molts dels seus aspectes.

- Art
  - 0: Un estil poc treballat.
  - 10: Molt atractiu visualment.

- Mecàniques
  - 0: Mecàniques molt simples i/o repetitives.
  - 10: Mecàniques divertides, varietat de mecàniques.

- Món
  - 0: Un món simple o únicament cal combatre.
  - 10: Un món complexa i que permet fer moltes coses.

- Història
    - 0: Sense història.
    - 10: Una història ben elaborada o amb un pes molt important.

	Dofus	**Wakfu**	**FFT**	**DQ Tact**	**DQ VIII**	**Kingdom Doom**
Originalitat	8	7	6.5	7	7	6.5
Art	8.5	9	6	9.5	8.5	7
Mecàniques	10	10	6	7.5	5	7.5
Món	9.5	9.5	3	5	7.5	7
Història	4	5	9.5	9	10	3

Taula 2 Quadre de paràmetres d'avaluació de l'estat de l'art

2.4 *Viabilitat del projecte*

2.4.1 *Viabilitat tecnològica*

- Software
    - Unity3D
    - Visual Studio
    - GIMP
    - GitHub

- Hardware
    - Un ordinador per treballar amb models 3D i amb el programa Unity3D

2.4.2 *Viabilitat econòmica*

- Software: Tot el software necessari pel desenvolupament del videojoc és gratuït

- Hardware: Ja es disposa del hardware necessari, així que el cost real és zero.
    - Preu de l'ordinador: 1.200 €

- Treballadors: El cost real de les hores de desenvolupament és zero, ja que el projecte es fa sense patrocinadors i el desenvolupament és personal.
    - Preu d'un programador junior (10 € / hora) treballant 8 hores diàries durant el temps que duri el desenvolupament.
    - Preu d'un dissenyador (16 € / hora) treballant 8 hores diàries durant el temps que duri el desenvolupament.

El cost del projecte en el temps que s'ha estat desenvolupant per aquest TFG seria aproximadament de 19.920 €, únicament per la demo que s'ha desenvolupat, s'han necessitat 18 setmanes (4 mesos) de treball.

2.4.3 *Viabilitat legal*

D'acord amb els estàndards PEGI, el videojoc pot ser jugat a partir dels 12 anys, de manera que s'haurà de mostrar l'etiqueta PEGI 12 per indicar quines persones hi poden jugar i, de manera indirecta, el contingut que es pot trobar dins el joc.

Tots els apartats artístics, models 3D, imatges, sons, etc. seran creats manualment sempre que es pugui, o bé es buscaran arxius sense drets d'autor. Sempre caldrà complir aquestes dos simples normes perquè el projecte sigui viable legalment.

2.4.4 *Conclusions*

El projecte és completament viable, gràcies a utilitzar software gratuït i comptar amb tot el hardware necessari. També tenim solucionat el tema dels treballadors, ja que és un projecte personal i no ens comportarà cap despesa, així que ja comptem amb la base essencial per iniciar el desenvolupament del videojoc.

Per la part de la viabilitat legal, caldrà utilitzar models 3D, imatges, sons, etc. sense drets d'autor o creats manualment. A part d'aquest inconvenient, caldrà col·locar l'etiqueta PEGI 12 per complir amb la normativa.

3. *Planificació*

El projecte serà desenvolupat per una única persona, per tant tot el volum de treball recau sobre ell. Serà important una bona planificació i organització, tant de les feines a fer, com del temps que ocuparan per dur a terme el joc en el temps planificat.

3.1 *Paquets de treball*

3.1.1 *Documentació*

Tasques: Documentació del joc, Documentació del codi.
Temporització: 18 setmanes (la documentació evolucionarà juntament amb el joc).
Fita: Memòria del projecte.

3.1.2 *Mecàniques de combats*

Tasques: Definir les mecàniques. Definir la relació entre elles. Definir la intel·ligència artificial dels enemics. Implementar les mecàniques. Implementar la intel·ligència artificial. Depurar les mecàniques i la intel·ligència artificial.
Temporització: 18 setmanes (La depuració obliga a allargar la temporització).
Fita: Correcta creació de les mecàniques de combat i la intel·ligència artificial.

3.1.3 *Mecàniques del món*

Tasques: Definir les mecàniques. Definir la relació entre elles. Definir els personatges. Implementar les mecàniques. Implementar els personatges. Depurar les mecàniques.
Temporització: 8 setmanes (La depuració obliga a allargar la temporització).
Fita: Correcta creació de les mecàniques del món.

3.1.4 *Mons del joc*

Tasques: Definir el mapa i les seves zones. Definir nivells. Definir els objectes. Creació del mapa. Creació dels objectes. Implementar els nivells.
Temporització: 4 setmanes.
Fita: Implementar un mapa amb nivells. Creació d'objectes del joc.

3.1.5 *Interfícies d'usuari sense impacte en la jugabilitat*

Tasques: Definir els menús del joc. Implementar els menús del joc.

Temporització: 1 setmana.

Fita: Creació dels menús del joc.

3.1.6 *Interfícies d'usuari dins el món del joc*

Tasques: Definir les interfícies del món del joc, Implementar les interfícies del món del joc.

Temporització: 4 setmanes.

Fita: Implementació de les interfícies d'usuari dins el món del joc.

3.1.7 *Art del joc*

Tasques: Definir l'art adient del joc. Recerca i creació de l'art del món i les interfícies, Aplicació de l'art del món i les interfícies.

Temporització: 6 setmanes.

Fita: Recerca, creació i implementació de l'art del joc.

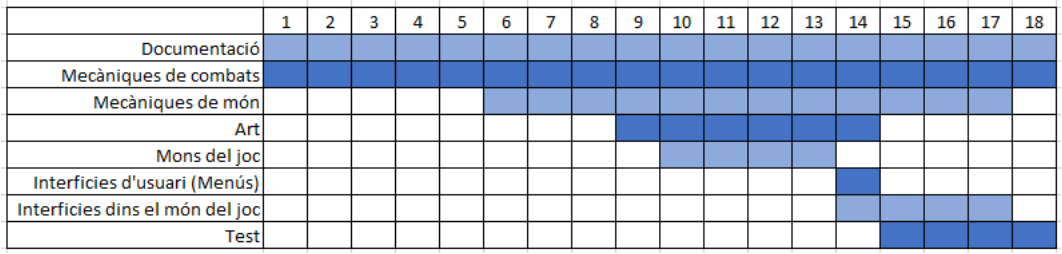
3.1.8 *Test*

Tasques: Recerca d'errors del joc final. Solucionar els errors del joc final.

Temporització: 4 setmanes.

Fita: Buscar i solucionar errors del joc final.

3.2 *Cronograma*



3.3 *Metodologia i Seguiment*

En aquest projecte totes les tasques recauen sobre una sola persona. Per tant, l'organització la defineix aquesta. En aquest cas, el sistema de treball ha constat amb un reconeixement previ de tot amb el que havia de comptar el videojoc final, posteriorment definint el que havia de contenir la demo per ser fidel al joc final i tenir una bona experiència de joc. Finalment, cada dues setmanes, es creava un llistat de tasques a fer. Aquestes no tenien una llargada de més de dos dies, un dia en la majoria dels casos. Un cop es completaven les tasques, es marcaven com a completades. D'aquesta manera es manté un historial de tasques creades i en quin transcurs de temps es van completar.

Aquest punt plantejarà les principals referències que es van tenir en compte per desenvolupar el joc i quins eren els plantejaments inicials. Tot això servirà per entendre les motivacions del projecte.

4.1 *Referències i conceptes*

Tal com s'ha especificat anteriorment en el document, aquest videojoc té unes referències molt clares A continuació, s'exposaran quins conceptes s'han tingut en compte, tant del gènere del videojoc com dels referents més propers.

4.1.1 *Gènere JRPG*

En el cas del gènere del joc, consta de dues parts, la part RPG, el que significa que el jugador ha de controlar les accions del personatge, com si fos ell mateix, ha de tenir una certa llibertat de moviment. També ha de poder evolucionar el personatge al llarg del temps, ja sigui amb experiència, amb armes i equipament o de qualsevol altre forma. Aquests aspectes estan plasmats en el videojoc, com el gènere indica, el jugador ha de prendre el rol del personatge principal i viure'n les experiències.

La segona part del gènere és la part tàctica, tots els jocs JRPG tenen una part d'estratègia, alguns més que d'altres, el que és indispensable són els combats per torns. Els principals videojocs del gènere, podríem dir que els creadors del gènere, tenen una part estratègica bàsica, és a dir, se solen limitar a una acció per torn, ja sigui atacar, defensar, utilitzar una poció, etc. com per exemple el Dragon Quest VIII que hem mencionat anteriorment.

Figura 8: Captura d'un combat de Dragon Quest VIII

Per altra banda, el nostre joc pertany a un altre grup dins el gènere, aquests tenen uns combats més complexos que permeten a l'usuari dur a terme diverses accions cada torn. Els jocs que més han inspirat en aquest aspecte del joc són el Dofus i el Wakfu, que tal com s'ha comentat anteriorment, el sistema de combats és molt similar. Aquests combats es desenvolupen a sobre d'unes caselles, com els escacs, en les quals ens movem lliurament i utilitzem atacs sobre aquestes. En el nostre cas tenim un màxim de moviments i manà cada torn que l'usuari l'administrarà com cregui més adient per sortir vencedor de cada lluita.

Figura 9: Captura d'un combat de Dofus

4.1.2 *Món del joc*

El joc compta amb un món obert on no es limita al jugador en els moviments, però la limitació la marquen els enemics, ja que si el jugador no és suficientment poderós, no podrà accedir a algunes zones, ja que no serà capaç de vèncer els enemics que hi trobarà. Aquest sistema de món és molt semblant al que ens podem trobar al joc Wakfu.

Figura 10: Captura d'un petit tros del mapa de Wakfu

4.1.3 *Missions i Equipament*

El sistema de missions i equipament que podem trobar al joc actualment no està directament referenciat a cap altre joc. En el cas de les missions, és un sistema de missions simples que es van generant aleatòriament. El sistema d'equipament compta amb una botiga on trobem les armes que es poden comprar. Aquestes armes tenen unes habilitats diferents que donaran al jugador una millora d'atac i/o defensa. Aquests dos aspectes no són pensats a partir d'un joc en concret, però en certa manera segueixen els estàndards dels jocs RPG.

4.1.4 *Escenari del joc*

El joc no serà supervisat, el jugador es mourà i completarà les missions com cregui més adient. El jugador es podrà moure dins un món relativament gran, però amb uns límits definits. La idea del joc és que en finalitzar la història, el jugador encara

trobi coses a fer pel món, així que el joc podríem dir que és rejugable fins a un cert punt, ja que en algun moment el jugador haurà completat el joc complet, però no acabarà juntament amb la història. Per completar el joc no hi ha una limitació de temps, aquesta limitació només es troba en els torns dels combats, que serveix per agilitzar les decisions de l'usuari i no allargar els combats innecessàriament.

4.2 *Entorns de treball*

Les eines utilitzades per completar el projecte son les següents:

- Implementació:
    - Motor: Unity3D
    - Entorn de desenvolupament: Visual Studio
- Software auxiliar:
    - Edició d'imatges: GIMP

Per desenvolupar el videojoc, en aquest cas, s'han utilitzat les eines exposades, totes utilitzades per una sola persona que ha estat l'encarregada de crear tots els aspectes que té un videojoc amb les característiques que hem explicat anteriorment. Els models 3D han estat utilitzats a través d'assets de Unity, cosa que ha permès administrar millor el temps pel desenvolupador. D'altra banda, tots els menús i el codi del projecte ha estat creat manualment, ja que tot i que el joc està inspirat en diversos aspectes d'altres videojocs, tot ha estat creat especialment per aquest projecte i el funcionament és completament personalitzat, així com l'aspecte dels menús i la distribució del mapa.

5. *Disseny del videojoc*

A continuació s'explicaran tots els aspectes del joc per separat, en aquest apartat es començarà a donar forma als plantejaments que hem fet en els punts anteriors.

5.1 *Narrativa*

La part narrativa del joc no està excessivament desenvolupada en la demo, a causa de les preferències que s'han tingut en el temps disposat. Tot i no tenir una narrativa molt forta, tot el joc segueix una història. L'ambientació implementada està pensada al voltant d'una història que transcorre en el lloc on es troba el personatge principal. Aquesta història tracta d'un regne de grans terres, ple de pobles en mig del bosc i amb un gran castell central on tots els pobles mantenen una gran comunicació amb el castell. El protagonista es troba en un dels pobles, on resulta que fa uns dies van perdre el contacte amb el castell i els ciutadans l'hi demanen si pot investigar que ha passat, ja que ell és el més jove i valent del poblat. Els ciutadans l'ajudaran amb el que puguin, però ells no saben res del que ha passat. Això ho haurà d'esbrinar per si sol.

5.2 *Personatges*

El personatge principal és un noi jove que viu en un poblat que ha perdut el contacte amb el castell més pròxim. Els ciutadans l'hi demanaran que descobreixi que ha passat. Al llarg de la història haurà de descobrir què és capaç de fer i com és de perillós el món en el qual viu. Caldrà que entreni i millori les habilitats així com l'equipament, tot per poder arribar a l'objectiu.

Figura 11: Captura del protagonista de Kingdom Doom

Al bosc que es troba als afores del poblat, juntament amb una cova que cal descobrir, s'hi troba una gran varietat dels enemics. Cada enemic té un comportament diferent. Tots els enemics que es trobarà el personatge principal en aquesta aventura són els següents:

- Slime: és un enemic molt simple, persegueix al jugador i només ataca quan està a una casella adjacent.

Figura 12: Captura d'un Slime de Kingdom Doom

- Aranya: aquest enemic té un atac molt dèbil i una vida molt baixa, però es mou com cap altre; si no pot atacar al jugador, s'escaparà d'ell.

Figura 13: Captura d'una Aranya de Kingdom Doom

- Metalon: existeixen tres tipus de Metalon diferents. El de color vermell té el moviment més limitat, però un atac més poderós i que a més a més desplaça al jugador. El Metalon de color lila compta amb una habilitat especial que resta manà al jugador. Aquesta habilitat l'utilitza si no pot atacar al jugador. Per últim, el Metalon de color verd té el mateix comportament que el de color lila, però la seva habilitat especial resta moviments al jugador.

Figura 14: Captura dels diferents Metalon de Kingdom Doom

També existeixen dos caps (bosses, anàlogues), els quals tenen unes habilitats especials que obliguen el jugador a prendre unes decisions diferents que en un combat normal. Aquests caps es troben en llocs concrets del mapa i recompensen al jugador amb molta experiència i diners. Els dos caps són els següents:

- Boletus: aquest cap es troba amagat al bosc que rodeja el poblat. La seva habilitat especial la pot utilitzar cada dos torns, i quan la tira resta tot el manà i moviments del jugador, obligant-lo a passar torn sense poder fer res. L'atac no és molt fort, però cada dos torns el jugador no serà capaç de fer res a causa de l'habilitat especial, per la qual cosa caldrà jugar intel·ligentment els torns per poder derrotar aquest enemic.

Figura 15: Captura del cap Boletus de Kingdom Doom

- Orc: el cap més temible de tots, ja que té un atac devastador. Si pot atacar al jugador, l'hi restarà una gran quantitat de vida, però no és capaç d'atacar cada torn, ja que ha de concentrar la seva força abans d'atacar. Amb aquest cap, el jugador caldrà que aprofiti el torn que el cap necessita concentrar-se per controlar les distàncies si no vol ser derrotat abans d'hora.

Figura 16: Captura del cap Orc de Kingdom Doom

5.3 *Espai i escenaris*

Dins el joc existeixen dos espais: l'espai del menú principal i el de joc. És el menú principal, on el jugador pot escollir si començar una nova aventura, continuar l'aventura en la qual es troba o bé sortir del joc.

Figura 17: Captura del menú inicial de Kingdom Doom

Per altra banda existeix l'espai de joc. En aquest espai és on es troben les aventures i on passarà tota l'acció. Dins aquest espai hi trobarem el menú de pausa, el menú de missions, la botiga i el menú final. Tots aquests menús es troben dins el mateix espai, únicament es mostren i posen en pausa el joc quan faci falta, sense canviar d'espai.

Figura 18: Captura del poble inicial de Kingdom Doom

El gràfic dels escenaris del joc quedaria de la següent manera:

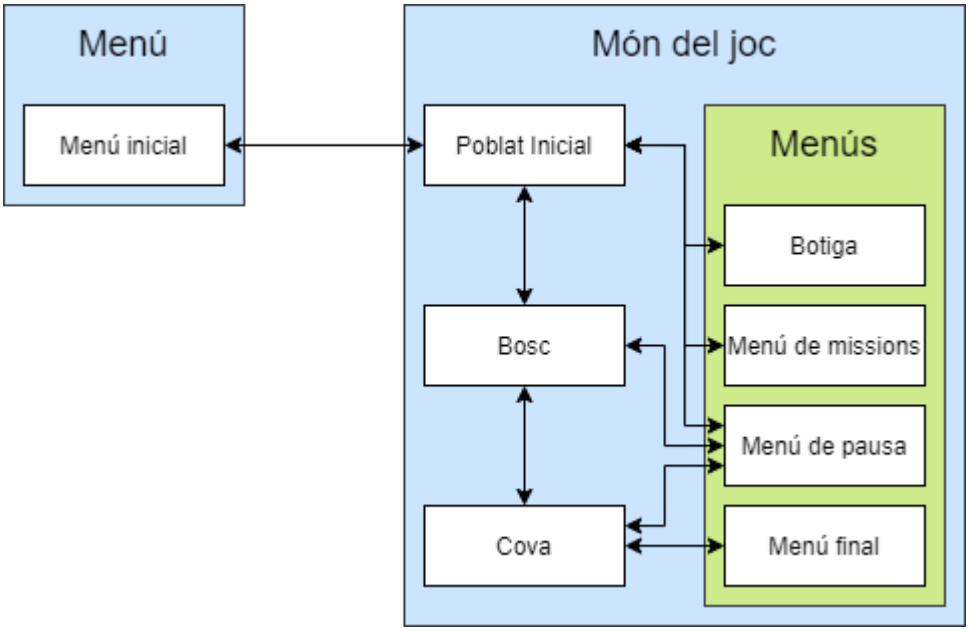


Diagrama 1: Escenaris de Kingdom Doom

5.4 *Mecàniques*

5.4.1 *Objectes*

Tots els objectes amb els quals compta el joc actualment són els següents:

- Player: Conté tots els atributs d'un personatge qualsevol i a més a més compta amb els atributs característics del personatge principal, com per exemple les monedes.

- Enemic: Conté tots els atributs d'un personatge qualsevol i, a més a més, compta amb els atributs característics dels enemics, com per exemple les monedes que donarà de recompensa al jugador si és derrotat, o el tipus d'enemic que es tracta.

- Agent: És l'encarregat de moure tots els personatges pel món, tant el jugador com els enemics es mouen pel món gràcies a l'Agent.

- Atac: Conté tota la informació i comportament de les habilitats que utilitza el personatge i els enemics.

- Equipament: Té tota la informació que requereixi qualsevol equipament del joc, sigui una arma o uns guants.

- Mànagers: Existeixen diferents mànagers dins el joc. Són els encarregats de marcar unes pautes i controlar els aspectes més essencials del joc, com per exemple el moviment del jugador dins el món o l'ordenació dels torns de combat.

5.4.2 *Accions*

El jugador pot dur a terme diferents accions dins el món del joc. En aquest apartat les mostrarem, juntament amb quin objecte es veu afectat per aquestes accions.

- Moviment pel món: Amb un simple clic en una part qualsevol del mapa, el jugador farà desplaçar el personatge corrents. Si al llarg del camí es troba amb algun enemic, entrarà en una lluita. Veure figura 19.

Figura 19: Captura del moviment del món de Kingdom Doom

- Interactuar amb la botiga i el cartell de missions: Quan el jugador es troba molt a prop del taulell de missions o la botiga, hi podrà interactuar amb el botó 'E'. Un cop ha interactuat s'obrirà un menú on trobarà diferents opcions on interactuar amb un simple clic. Veure figura 20.

Figura 20: Captura interacció amb la botiga de Kingdom Doom

- Activar i aplicar habilitats: Per guanyar un combat és necessari utilitzar les habilitats del jugador. Aquestes s'activen fent clic a la interfície d'usuari que es troba a la part inferior de la pantalla. En seleccionar una habilitat, es ressaltaran les caselles del mapa amb les que es pot utilitzar aquesta habilitat. Per utilitzar l'habilitat caldrà fer clic a una de les caselles. Veure figura 21

Figura 21: Captura utilització habilitats de Kingdom Doom

- Activar i aplicar moviment: En un combat el jugador té un màxim de moviments cada torn. Amb un clic al botó de la part inferior esquerra de la interfície d'usuari, es podran utilitzar els moviments. En fer clic, es ressaltaran totes les caselles on sigui possible moure's, i després en fer un segon clic a una casella donada, el jugador caminarà fins a arribar-hi. Veure figura 22.

Figura 22: Captura utilització del moviment dels combats de Kingdom Doom

5.4.3 *Reptes*

Els reptes del joc els podríem separar en dos tipus, els que es troben dins els combats i els que es troben dins el món. En aquest apartat els definirem i els connectarem entre si. Veure el diagrama 2.

Reptes del món:

- Descobrir el mapa: Desplaçant al personatge pel món descobrirem noves zones del mapa. Cada zona té un cap que haurem de derrotar, i podem avançar en el joc.

- Entrar en combat: Per entrar en combat només cal moure al personatge a una o dues caselles de distància d'un enemic. Quan el personatge es trobi a prop, entrarà en combat.

- Completar missions: Per poder completar missions cal recollir missions al taulell de missions. Posteriorment caldrà buscar els enemics que ens demana derrotar la missió. Quan s'hagin derrotat tots els enemics, caldrà tornar al cartell i demanar la recompensa. Veure figura 23.

Figura 23: Captura taulell de missions de Kingdom Doom

- Comprar i equipar armes: Primer de tot caldrà aconseguir diners derrotant enemics en combats i completant missions. Quan es tinguin els diners suficients, la botiga del poblat ens permetrà comprar les armes. Un cop adquirides, les podem equipar des de la botiga en qualsevol moment.

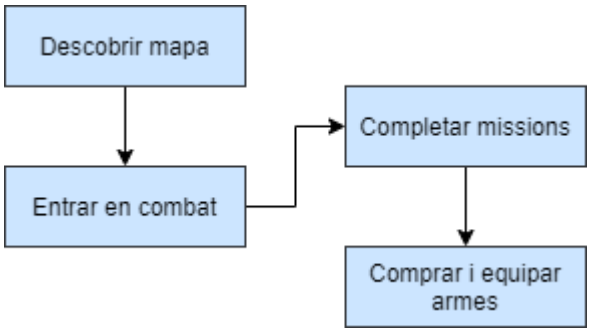


Diagrama 2: Jerarquia de reptes del món

Reptes dels combats (Veure diagrama 3):

- Moure al personatge: A la interfície d'usuari trobarem un botó per al moviment del personatge. Caldrà fer un clic i d'aquesta manera mostrarà les possibilitats, ressaltant les caselles on es pugui desplaçar el personatge. El moviment es farà efectiu en clicar una de les caselles ressaltades que apareixen. Veure figura 24.

Figura 24: Captura moviment dels combats de Kingdom Doom

- Atacar, curar i utilitzar habilitats: Aquests tres reptes es fan a través dels atacs, és a dir, el funcionament és el mateix. Els diferents atacs dels quals disposa el jugador depenen de les armes i equipament que porta el jugador. Per utilitzar un atac, el funcionament és el mateix que per moure el jugador: selecciona un atac i es mostren les caselles possibles. El jugador, en passar el ratolí per les caselles, veurà l'àrea d'afecte i, en seleccionar una casella, es llançarà l'atac. Veure figura 25.

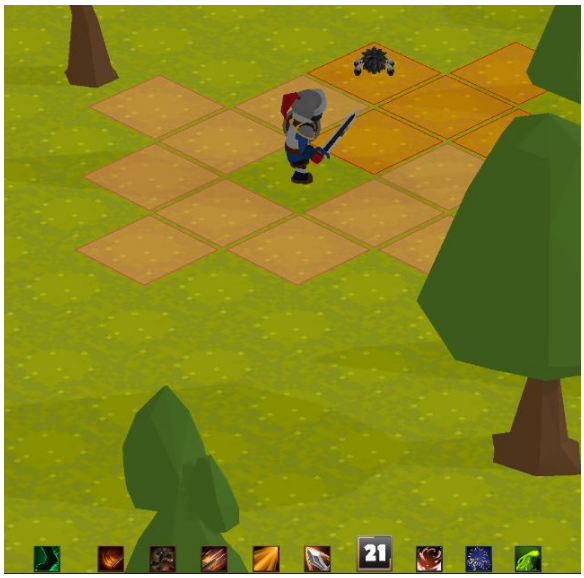


Figura 25: Captura atacs dels combats de Kingdom Doom

- Passar el torn: En el moment que és el torn del jugador, a la part central inferior de la pantalla, es mostra un compte enrere. Quan el jugador hagi finalitzat totes les accions, podrà clicar el botó on es mostra el temps restant, ometent el temps i finalitzant el torn.

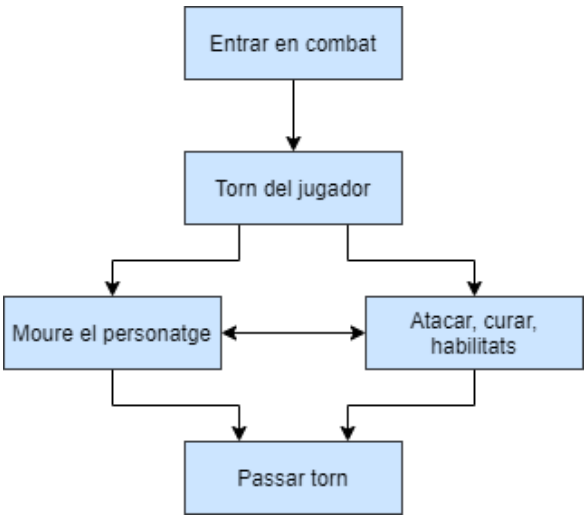


Diagrama 3: Jerarquia de reptes dels combats

5.5 *Economia del joc*

L'economia del joc està dividida en dos tipus, l'economia dins de cada combat i l'economia del món. Les dues tenen unes funcions molt diferents, i es relacionen entre si a través de l'equipament que porti equipat el jugador i l'experiència que ha guanyat al llarg del joc.

5.5.1 *Recursos*

Recursos del món:

- Experiència i nivell: L'experiència del jugador determinarà el seu nivell i aquest n'augmentarà la seva vida i atac. L'experiència s'aconsegueix lluitant i completant missions.
- Monedes: Les monedes serveixen per comprar equipament més poderós. Les monedes s'aconsegueixen lluitant i completant missions.

Recursos de combat:

- Mana: L'equipament que tingui equipat el jugador determinarà quines habilitats pot utilitzar en combat. Aquestes habilitats necessiten el mana per poder ser utilitzades. El jugador té un mana establert per l'equipament i nivell. Cada torn es reiniciarà aquest mana. Al llarg d'un torn el jugador pot gastar i augmentar el manà amb les habilitats.

- Moviments: El funcionament dels moviments és el mateix que el manà, el jugador té uns moviments establerts cada torn que els pot gastar o augmentar. El jugador només es pot desplaçar a les cel·les adjacents, mai en diagonal.

- Vida: La vida del jugador ve determinada pel nivell i equipament. Fora de combat, si al jugador l'hi falta vida, la recuperarà a poc a poc. Un cop en combat, ja no es recuperarà la vida, caldrà utilitzar les habilitats per poder curar la seva vida. La vida té un valor màxim i mai es pot superar, en arribar la vida a 0 el jugador es desmaiarà i tornarà al poblat per recuperar-se.

- Atac: L'atac del jugador el determina l'equipament. Aquest afecta directament a les habilitats que utilitza dins els combats. L'atac dels enemics dependrà directament del tipus d'enemic.

- Defensa: La defensa del jugador la determina l'equipament, aquesta afecta directament a la disminució del dany rebut de les habilitats dels enemics. La defensa dels enemics dependrà directament del tipus d'enemic.

5.5.2 *Entitats*

Personatge (Valors inicials)

Experiència	0
Nivell	1
Monedes	0
Mana	6
Moviments	3
Vida	30
Atac	0
Defensa	0

Taula 3 Quadre d'entitats del personatge

Cal tenir en compte que el mana i els moviments varien al llarg del combat, així com la vida que també es veu afectada fora dels combats i pel nivell del personatge. L'atac i la defensa seran definits per l'equipament que es vagi comprant el jugador.

Enemic (Valors inicials)

Experiència	3-40
Nivell	1
Monedes	1-20
Mana	4-6
Moviments	3-8
Vida	3-25
Atac	0-50
Defensa	0-50

Taula 4 Quadre d'entitats dels enemics

L'experiència i monedes dels enemics no defineixen els atributs propis, sinó que defineixen el valor del recurs que rebrà el jugador si aconsegueix derrotar-los. Tots els atributs dels enemics venen donats depenent del tipus, i també es veuran modificats segons el nivell. En aquest cas, a la taula 4, es mostren els atributs per un enemic a nivell 1.

5.5.3 *Funcions*

Source: L'experiència i les monedes augmenten de la mateixa manera, en finalitzar un combat o en finalitzar una missió. Per altra banda, tant el mana com els moviments dins un combat, com la vida, poden augmentar dins un combat fent ús de les habilitats que atorga l'equipament del jugador. En el cas que el jugador no tingui tota la vida en finalitzar un combat, la recuperarà a poc a poc mentre no entri en una altra lluita.

Drain: Quan el personatge o els enemics fan ús de les habilitats, necessiten una quantitat de mana per utilitzar-la, de tal manera que perden mana en tirar una habilitat. Algunes d'aquestes habilitats serveixen per danyar a l'oponent, per tant, fan perdre vida a l'oponent. Els personatges també es poden moure dins el terreny de joc seguint les caselles. Quan un personatge es mou, gasta els punts de moviment que tenia per aquell torn.

Converter: En el moment que el personatge principal ha aconseguit l'experiència suficient per pujar de nivell, aquesta evoluciona a un nivell nou automàticament, és a dir, el personatge guanya un nivell, però reinicia l'experiència.

Trader: En el poblat hi ha una botiga on el jugador podrà intercanviar les monedes que tingui per equipament, d'aquesta manera, intercanviarà les monedes per equipació que l'hi proporcioni millors habilitats i atributs.

5.5.4 *Representació de les interfícies*

Els recursos que hem definit anteriorment, es veuen representats per la interfície d'usuari de diferents maneres. L'experiència, els nivells, la vida, les monedes, el mana i els moviments dins de combat estan representats en tot moment a la part inferior dreta de la pantalla, tant si el jugador es troba dins un combat o no.

Figura 26: Captura de la interfície d'usuari del Kingdom Doom

La informació més important de l'enemic, és a dir, la vida, el manà, els moviments i el nivell, es mostren quan el ratolí se situa sobre l'enemic dins un combat.

Figura 27: Captura de la interfície d'usuari del Kingdom Doom

L'experiència i les monedes que donen de recompensa per completar una missió es mostraran tant en el cartell de missions com en la interfície d'usuari a la part superior dreta de la pantalla, on es pot veure com evoluciona la missió que està activa.

Figura 28: Captura de la interfície d'usuari del Kingdom Doom

Quan un personatge utilitza una habilitat, qualsevol recurs que es vegi afectat es mostrarà a través d'uns números flotants sobre el personatge afectat. Per exemple, si el jugador utilitza un atac i gasta mana, es veurà el cost de mana amb un número flotant sobre el jugador. Si aquest atac ha afectat un enemic i l'hi ha causat mal, es veurà un número indicant el dany rebut. Aquests números venen acompanyats per un color i una icona que ajuda a identificar a quin recurs fan referència.

Figura 29: Captura de la interfície d'usuari del Kingdom Doom

5.6 *Interfícies*

El joc compta amb moltes interfícies, en aquest apartat es definiran totes i cada una i s'explicarà el funcionament.

5.6.1 *Menú inicial*

La primera interfície que ens trobem en iniciar el joc és el menú inicial. Aquest menú és molt simple, ja que només compta amb tres opcions: iniciar una nova aventura, continuar l'aventura guardada o sortir del joc.

Figura 30: Captura del menú inicial del Kingdom Doom

5.6.2 *Menú de pausa*

El menú de pausa es pot obrir en qualsevol moment del joc. Aquest menú és encara més simple que el menú inicial, ja que només ens permet reprendre el joc tal com l'havíem deixat, o tornar al menú inicial.

Figura 31: Captura del menú de pausa del Kingdom Doom

5.6.3 *Menú de missions i botiga*

El menú de la botiga ens mostra tota l'equipació que es pot comprar o ja està comprada. Des d'aquest menú podrem comprar nova equipació i equipar la que ja s'ha comprat anteriorment.

Figura 32: Captura de la botiga del Kingdom Doom

Per activar una nova missió, caldrà anar al taulell de missions. Aquest taulell ens permetrà veure tota la informació de les missions que hi ha actualment, així com activar una de nova, completar la missió activa o eliminar una missió. El taulell permet com a màxim tres missions, en eliminar una o completar-la, deixarà un espai perquè aparegui una missió nova.

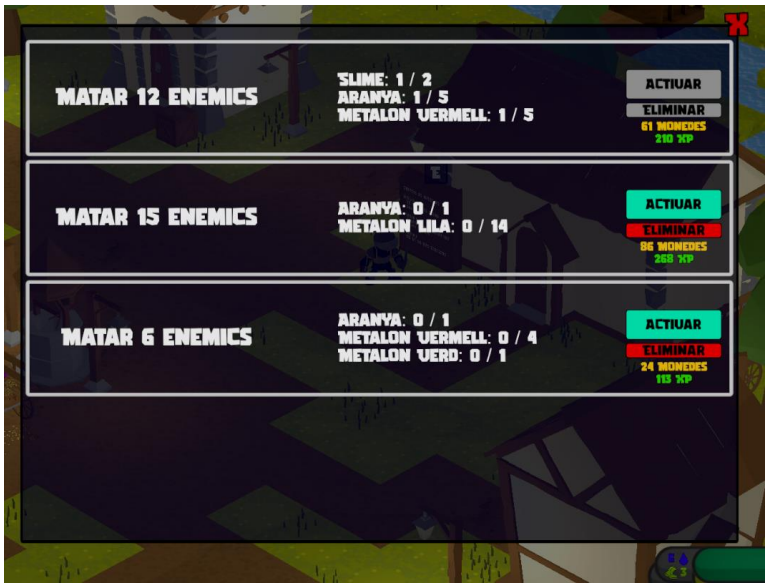


Figura 33: Captura del taulell de missions del Kingdom Doom

5.6.4 *Interfície del món*

La interfície que es veu en tot moment al joc ens dona la informació més essencial del personatge. A la part inferior dreta es pot consultar la vida, l'experiència, el nivell, les monedes, el mana i els moviments de combat. D'altra banda, a la part superior dreta de la pantalla, s'hi troba un desplegable que mostra tota la informació de la missió que estigui activa en aquell moment.

Figura 34: Captura de la interfície del món de Kingdom Doom

Quan el jugador viatgi pel món, es trobarà diferents enemics. En passar el ratolí per sobre d'un d'aquests, es mostrarà la informació del grup d'enemics en el que es troba. En aquesta interfície es pot veure el nom dels enemics, el nivell i en cas que tinguin una estrella, això significa que és un dels caps.

Figura 35: Captura de la interfície del món de Kingdom Doom

5.6.5 *Interfície de combat*

En entrar en combat, es mostra, a la part inferior de la pantalla, un botó per activar el moviment del jugador, a la part esquerra, totes les habilitats que tingui actives el jugador i un temporitzador a la part central. Els primers trenta segons d'entrar en combat, serveixen per col·locar als personatges i mostrar al jugador la situació del combat. Si el jugador vol finalitzar aquest torn previ al combat, caldrà que premi el botó que es troba sobre el temporitzador amb el text "START". D'aquesta manera es finalitzarà aquest estat i s'iniciaran els torns de lluita.

Per moure al personatge o utilitzar una habilitat caldrà clicar al botó que es desitgi. Posteriorment se'n mostrarà l'abast i caldrà seleccionar la casella en la qual volem fer efectiva l'acció.

Totes les accions es veuran reflectides en la interfície situada a la part inferior dreta que hem vist en l'apartat d'interfícies de món.

Per finalitzar un torn abans d'esgotar el temps, prement el botó on es troba el temporitzador esgotarà el temps i passarà directament al següent torn.

Figura 36: Captura de la interfície de combat de Kingdom Doom

En passar el cursor per sobre d'una de les caselles activades prement el botó de moviment o d'una habilitat, es mostrarà o bé el camí que seguirà el jugador per arribar a la casella, o l'àrea d'acció de l'habilitat.

Quan el jugador es troba en combat i passa el cursor per sobre d'un enemic, es mostraran les dades més importants de l'enemic. Les dades són: la vida, el manà, els moviments, el nivell, el nom i l'estrella ens indica que és un cap.

Figura 37: Captura de la interfície de combat de Kingdom Doom

Abans d'activar una habilitat, en situar el ratolí sobre el botó de l'habilitat, es desplegarà una petita interfície amb les dades de l'atac. Aquestes són: la descripció, el mana que demana, l'atac, la cura, quantes caselles empenyerà, l'abast de l'atac, el tipus d'atac (en àrea o en línia), la visió de l'atac (si travessa parets o no), l'àrea d'acció, el tipus d'àrea (en àrea o en línia), el mana i passos restats a l'objectiu.

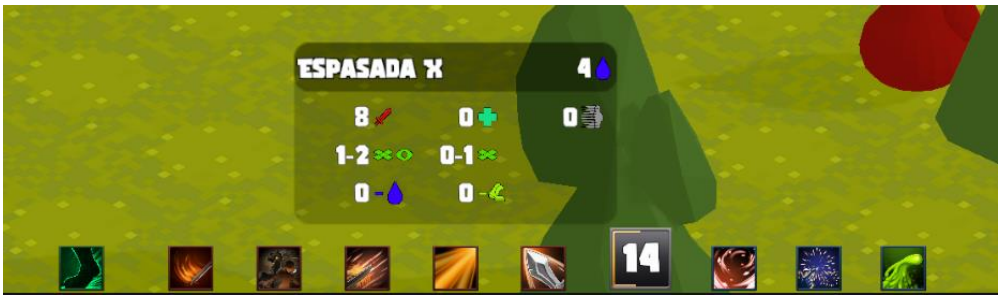


Figura 38: Captura de la interfície de combat de Kingdom Doom

Quan un personatge fa efectiu un moviment, atac, cura o habilitat, es mostra una animació de l'atac, a més de mostrar uns números flotants amb una icona al costat i un color concret, que fan referència als recursos de l'economia del joc que es vegin afectats per l'acció. Els recursos que poden ser mostrats són: la vida restada (atacs), la vida augmentada (cures), el mana i els moviments.

Figura 39: Captura de la interfície de combat de Kingdom Doom

5.6.6 *Menú final*

En el moment que el jugador venç el cap més fort, es mostra el menú final del joc, informant el jugador que ha completat tots els reptes i que pot finalitzar el joc, o bé continuar l'aventura i desbloquejar tot l'equipament o derrotar tots els caps en cas de no haver-ho fet anteriorment. Veure figura 40.

Figura 40: Captura del menú final del Kingdom Doom

5.6.7 *Principis de Mayer*

1. Principi de coherència: El tutorial inicial pot ser una mica extens, ja que els combats contenen moltes variables que el jugador ha de conèixer. Generalment, la informació que es dona al jugador és breu i se sol mostrar només la informació important.

2. Principi de senyalització: La informació important ressalta respecte al color de fons. A més a més, sempre té una bora de color negre per ajudar a l'usuari a visualitzar-ho més fàcilment.

3. Principi de redundància: Per regla general, el joc conté poc text, tot el text que es mostra és informatiu i estrictament necessari per a la informació que es dona. El tutorial és l'única part més extensa i amb una explicació més completa, però en cap cas redundant i innecessària.

4. Principi de contigüitat espacial: Quan es mostra una informació es fa dins uns espais definits, dins uns blocs de text que ressalten sobre el món, de tal manera que tota la informació queda concentrada dins el bloc. Si els texts no estan en blocs, apareixen en el lloc concret on afecten. Per exemple, en un combat, si un personatge gasta mana per curar-se, apareixeran dos valors sobre el personatge indicant el cost de mana i la cura.

5. Principi de contigüitat temporal: Sempre que es mostra una informació, sigui un text o un simple número, van acompanyats o per un títol, una il·lustració o icona que indica a què fa referència.

6. Principi de segmentació: Tota mena d'informació que se mostra es centra únicament en el que interessa a l'usuari en cada moment. És a dir, quan l'usuari té el ratolí sobre un enemic, només veurà informació de l'enemic per pantalla,

7. Principi de pre-entrenament: Com s'ha vist en l'apartat d'interfícies d'usuari, abans de llançar una habilitat, es mostra tot el funcionament en un bloc flotant. Per tant, el jugador sabrà com funciona l'atac abans d'utilitzar-lo. En el tutorial es mostraran diferents aspectes del joc que caldrà que el jugador conegui abans de començar, però hi haurà aspectes que caldrà que descobreixi per si mateix.

8. Principi de modalitat: Gran part de la informació és necessàriament en format text, ja que cal mostrar els números de monedes, experiència, dany, cura, mana, moviments, etc., que demana l'equipament, les habilitats i les missions. Per agilitzar la comprensió del jugador, tots aquests textos van acompanyats d'un color característic i una icona, de tal manera que amb una mirada ràpida és fàcil d'entendre tota la informació.

9. Principi de multimèdia: Tal com s'ha dit al llarg d'aquest punt, els textos venen acompanyats d'icones i un color característic per una percepció més ràpida i un major enteniment.

10. Principi de personalització: Totes les explicacions o textos del videojoc són en un to casual i a vegades amb un toc humorístic. En cap cas formal però tampoc amb males formes.

11. Principi de veu: En un principi no està planejat utilitzar veus per a ensenyar coses al jugador. Quan cal ensenyar alguna cosa a l'usuari, es farà a través de text, imatges i/o icones.

12. Principi d'imatge: Quan cal ensenyar una informació al jugador que no coneix, com per exemple el tutorial, que donarà a conèixer el funcionament del joc al jugador, cada nou aspecte estarà acompanyat d'una imatge real del joc que mostri el que s'està explicant.

5.7 *Ambients*

La demo se situa en el poblat principal, on el jugador podrà agafar missions i completar-les i comprar noves armes amb diferents habilitats, així com millorar la seva força i defensa per poder enfrontar-se als enemics que es trobarà pel món.

Figura 41: Captura del poblat del Kingdom Doom

Al voltant del poblat, s'hi troba un bosc, en el qual ens podrem trobar els Slimes i les Aranyes, juntament amb el cap Boletus. Aquest bosc no és molt extens, ja que hi ha un riu que ens barrarà el pas o una gran multitud d'arbres que no ens deixaran passar.

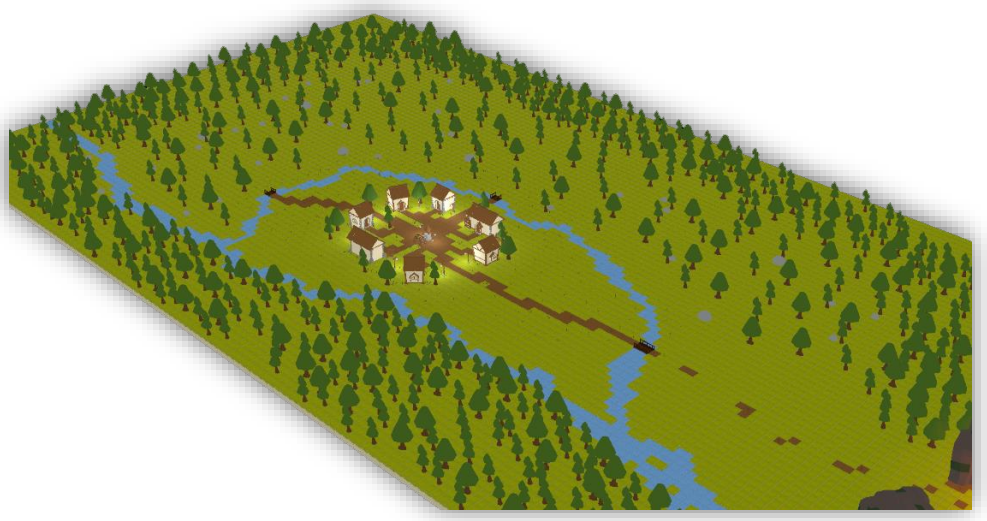


Figura 42: Captura del bosc del Kingdom Doom

A la sortida est del poblat es troba una cova, la qual està plena de torxes que il·luminen el camí. Aquestes torxes tenen una animació que varia la intensitat de la llum com si tinguessin un foc encès. En aquesta cova ens trobarem un petit rierol interior i una cascada. També hi podrem trobar els enemics anomenats Metalon, juntament amb un cap que és l'Orc.

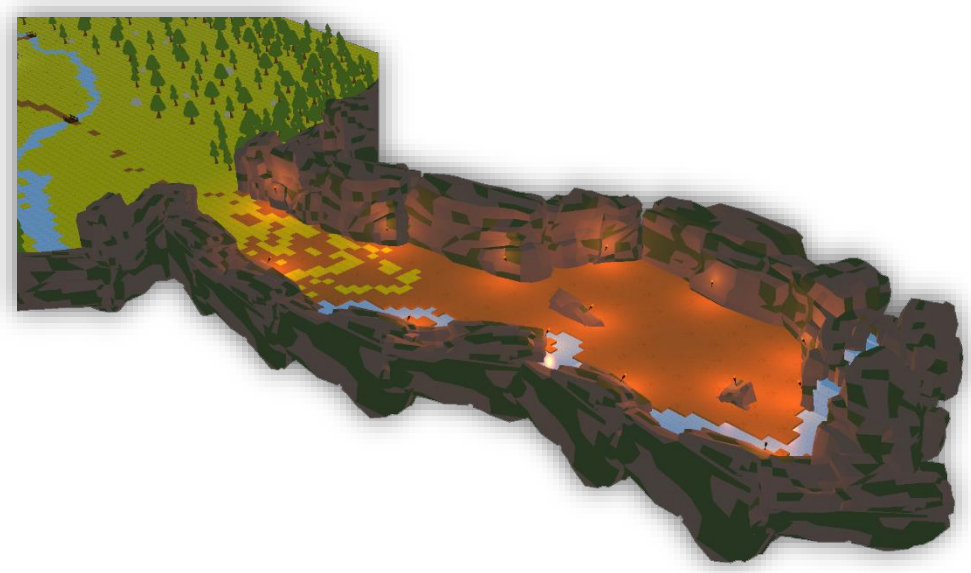


Figura 43: Captura de la cova del Kingdom Doom

5.8 *Elements a desenvolupar*

El desenvolupament es pot separar en diferents grans blocs que funcionen completament per separat, però tots tenen impacte entre si. Cada un dels blocs té diferents elements imprescindibles per al funcionament. Els blocs i elements desenvolupats són els següents:

- Bloc del món del joc
    - Creació de mapa
    - Moviment dels personatges
    - Interacció amb la botiga i taulell de missions
    - Creació aleatòria de grups d'enemics

- Bloc dels combats:
    - Organització dels torns de combat
    - Atributs dels personatges i habilitats
    - Moviment dels personatges
    - Habilitats dels personatges
    - Intel·ligència artificial dels enemics
    - Animacions de les habilitats

- Bloc de les interfícies:
    - Creació dels menús
    - Creació de les interfícies del món
    - Creació de les animacions de les interfícies

5.9 *Adaptabilitat*

El jugador començarà l'aventura amb un petit tutorial del funcionament bàsic del món, la botiga, les missions i els combats. Amb aquest coneixement inicial podrà començar a descobrir el mapa com vulgui o com els enemics l'hi permetin, ja que es trobarà a nivell 1 amb una arma molt bàsica, així que hi haurà enemics que seran molt difícils o impossibles de vèncer de primeres. El jugador haurà de pujar de nivell i aconseguir diners per millorar la seva força i equipament dins els combats i d'aquesta manera vèncer tots els enemics i, finalment, vèncer el cap que es trobarà dins la cova.

5.10 *Flowchart*

El diagrama de flux del videojoc comença des del menú inicial, on iniciarem o continuarem el món del joc, on el jugador es mourà lliurement entre el poblat, el bosc i la cova. El jugador podrà accedir al menú de pausa en tot moment mentre es troba dins el món del joc, però els combats únicament es troben al bosc i a la cova. Per entrar al menú final cal derrotar al cap més fort del joc, que es troba dins la cova.

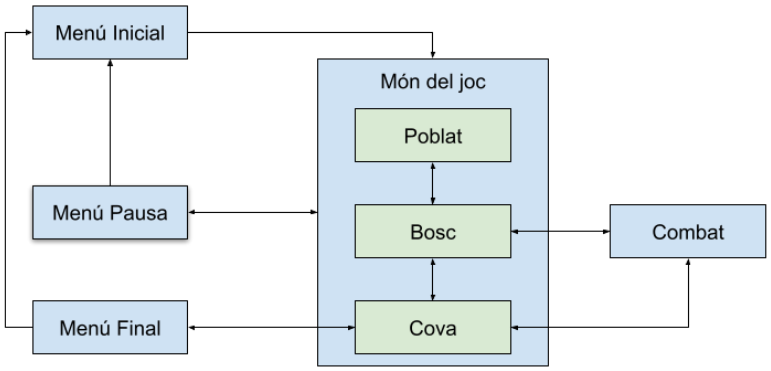


Diagrama 4: Flowchart del Kingdom Doom

Per desenvolupar un videojoc no només es necessiten coneixements de programació o d'art, és important saber solucionar problemes i prendre decisions per agilitzar el desenvolupament tot el que sigui possible. En aquest apartat es descriurà amb més detall el desenvolupament, es mostraran i s'explicaran parts del codi i decisions de la implementació amb el motor gràfic Unity que puguin ser interessants i essencials del joc.

6.1 *Organització*

Una bona organització és important en qualsevol projecte i per aquest motiu s'ha mantingut una estructura de carpetes, fitxers i objectes en els diferents aspectes de Unity. L'estructura de les carpetes que contenen tots els fitxers del joc és la mostrada a la figura 43.

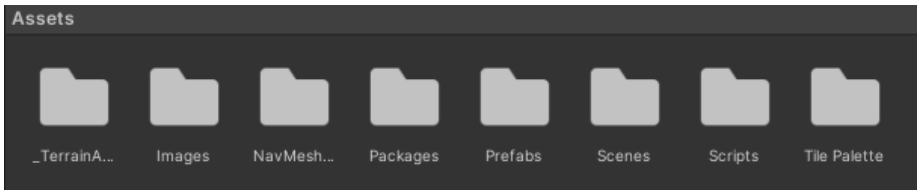


Figura 43: Estructura de carpetes del projecte de Unity

A mesura que un projecte creix és important saber on trobar el que es necessita el més ràpid possible, per això s'han creat les carpetes Imatges, Prefabs, Scenes i Scripts que es veuen a la Figura 43. Aquestes carpetes contenen tots els fitxers que s'han creat a mà. Per altra banda, tenim la carpeta Packages que conté totes les llibreries externes. La resta de carpetes s'han creat soles des del motor Unity. L'estructura dels objectes que componen el món del joc és la mostrada a la figura 44.

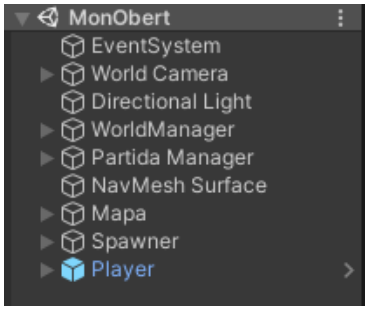


Figura 44: Estructura d'objectes projecte de Unity

Una escena de Unity està composta enterament d'objectes que es poden posar uns dins els altres com si fossin carpetes. En cas d'aplicar un canvi al pare, afectarà el fill, això és molt important per entendre la Figura 44. L'estructura és molt important pel funcionament de cada un dels objectes, per aquest motiu, tal com es mostra a la Figura 44, l'objecte Player, per exemple, es troba a l'arrel del projecte, ja que no depèn de res més, o el Mapa, que conté tots i cada un dels objectes que creen el mapa del món.

6.2 *Objectes*

Els objectes del joc s'han definit anteriorment en l'apartat 5.4.1, en aquest apartat es definiran els atributs i per a què serveixen. Per altra banda, les corresponents funcions es troben explicades als apartats 6.3, 6.5 i 6.6.

- Atributs Personatge: Atributs de qualsevol personatge.
    - Atributs
        - Nivell: Determina el nivell del personatge.
        - Experiencia: Nombre d'experiència actual.
        - VidaInicial: Vida a nivell 1.
        - VidaTotal: Vida màxima, tenint en compte la vida inicial i el nivell.
        - VidaActual: Vida actual.
        - ManaTotal: Mana màxim.
        - ManaActual: Mana actual.
        - MovimentsTotal: Moviments màxims.
        - MovimentsActual: Moviments actuals.
        - Estats: Guarda els estats que té el personatge que s'hagin aplicat amb les habilitats.

- Agent: És l'encarregat del moviment dels personatges pel món.
    - Atributs
        - Velocitat: Velocitat de moviment.
        - Target: Posició a la qual cal arribar.
        - Rotacio: Rotació del personatge.

- Player: S'encarrega del comportament del personatge del jugador.
    - Atributs
        - Arma: Arma equipada.
        - AtributsPersonatge: Objecte Atributs Personatge.
        - Monedes: Monedes que té el jugador.
        - Actiu: Determina el torn del jugador en combat.
        - EnMoviment: Està o no en moviment.
        - Velocitat: Velocitat de moviment.
        - AccioActual: Guarda l'acció que ha activat el jugador.

- Enemic: S'encarrega del comportament dels enemics.
    - Atributs
        - Nom: Nom de l'enemic.
        - Boss: Diu si és o no un cap.
        - AtributsPersonatge: Objecte Atributs Personatge.
        - Actiu: Determina el torn de l'enemic en combat.
        - EnMoviment: Està o no en moviment.
        - Velocitat: Velocitat de moviment.
        - Monedes: Monedes de recompensa al jugador en ser derrotat.
        - Atac: Valor d'atac.
        - Defensa: Valor de defensa.
        - Tipus: Determina si és agressiu o passiu.
        - Atacs: Habilitats que té l'enemic.

- Atributs Atac: Les habilitats, independentment del tipus, s'anomenen atacs.
    - Atributs
        - Descripcio: Descripció de l'atac
        - Mana: Mana que gasta l'atac en ser utilitzat.
        - Damage: Dany que causa l'atac.
        - Heal: Cura que causa l'atac.
        - RangAtac: Rang de cel·les de visió de l'atac.
        - TipusAtac: Tipus del rang de l'atac.

        - RangArea: Rang de l'àrea d'acció de l'atac.
        - TipusArea: Tipus de l'àrea d'acció de l'atac.
        - Habilitats: Habilitats especials que aplica l'atac.

- Atributs Equipament: Atributs de qualsevol equipament.
    - Atributs
        - Descripcio: Descripció de l'equipament.
        - Tipus: Tipus d'equipament.
        - Atac: Atac que proporciona al personatge.
        - Defensa: Defensa que proporciona al personatge.
        - Atacs: Atacs que permet utilitzar al personatge.

- Missio: S'encarrega de la gestió de les missions.
    - Atributs
        - EnemicsNecessita: Enemics necessaris per completar la missió.
        - EnemicsActuals: Enemics derrotats actualment.
        - Monedes: Monedes de recompensa.
        - Experiencia: Experiència de recompensa.

6.3 *Funcions de l'economia del joc*

Les funcions de l'economia del joc s'han definit anteriorment en l'apartat 5.6.1. En aquest apartat es definiran les entitats afectades i les funcions dels scripts encarregades de controlar els valors de l'economia del joc.

Source

- Entitats afectades:
    - Personatge. Experiència
    - Personatge. Monedes
    - Personatge. Mana
    - Enemic. Mana
    - Personatge. Moviments

- Enemic. Moviments
- Personatge. Vida
- Enemic. Vida

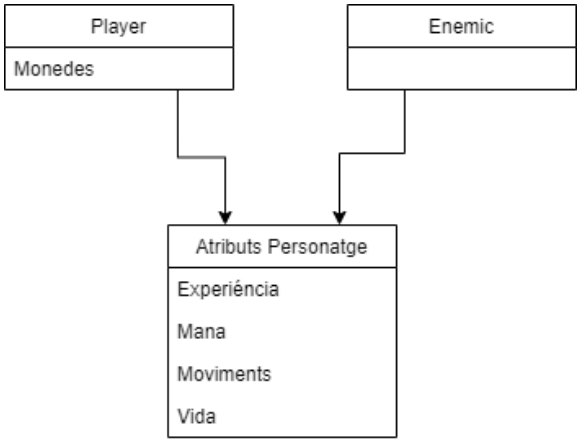


Diagrama 5: Objectes afectats pels source

- Funcions:

Funció que recull l'experiència guanyada.

```
Funció TakeExperiencia(ulong xp)
    Personatge.Experiència += xp
    // Posteriorment té en compte el nivell
```

Funció que recull les monedes guanyades.

```
Funció TakeMonedes(int monedes)
    Personatge.Monedes += monedes
```

Funció que suma el manà.

```
Funció TakeMana(int mana)
    Personatge.Mana += mana
```

Funció que suma els moviments.

```
Funció TakePassos(int moviments)
    Personatge.Moviments += moviments
```

Funció que puja la vida.

```
Funció TakeHeal(int heal)
    Personatge.Vida += heal
```

Drain

- Entitats afectades:
    - Personatge. Mana
    - Enemic. Mana
    - Personatge. Moviments
    - Enemic. Moviments
    - Personatge. Vida
    - Enemic. Vida

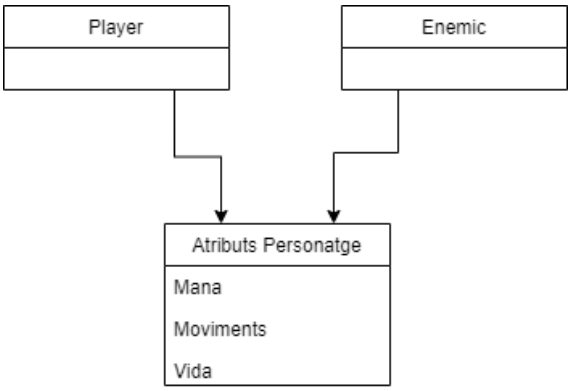


Diagrama 6: Objectes afectats pels drain

- Funcions:

Funció que resta el manà.

```
Funció TakeRestaMana(int mana)
    Personatge. Mana -= mana
```

Funció que resta els moviments.

```
Funció TakeRestaPassos(int moviments)
    Personatge. Moviments -= moviments
```

Funció que a partir del dany enemic i la defensa del personatge, calcula i resta la vida.

```
Funció TakeDamage(int damage)
    damage -= (damage * defensa / 100) // Calcula el damage
    tenint en compte la defensa
    Personatge. Vida -= damage
```

Converter

- Entitats afectades:
    - Personatge. Experiència
    - Personatge. Nivell

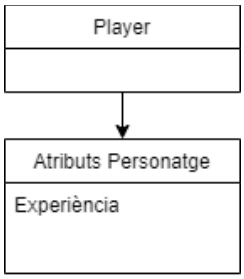


Diagrama 7: Objectes afectats pels converter

- Funcions:

Funció que en recollir l'experiència guanyada puja de nivell, si cal.

```
Funció TakeExperiencia(ulong xp)
    Personatge. Experiència += xp
    Si (Personatge. Experiència >= experienciaNecessaria)
        Personatge. Experiència -= experienciaNecessaria
        Personatge. Nivell += 1
```

Trader

- Entitats afectades:
    - Personatge. Monedes

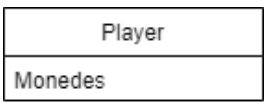


Diagrama 8: Objectes afectats pels trader

- Funcions:

Funció que en comprar equipament en resta els diners al personatge.

```
Funció CompraEquipament(int idEquipament)
    Si (Personatge. Monedes >= preu)
        Personatge. TakeMonedes(-preu)
```

6.4 *Entorn*

El mapa del joc està creat per blocs, cadascun és igual de gran que una casella. Els altres objectes que es troben al món no tenen aquesta relació amb les caselles. Depenent de la seva mida i posició s'han adaptat les caselles que ocupa cada objecte del mapa. L'estructura del mapa permet una millora del rendiment del joc, ja que s'ha creat un 'Occlusion Culling' que consisteix a activar únicament els objectes que es troben dins la visió de la càmera. Això fa que el joc deixi de calcular si ha de mostrar o no per pantalla certs objectes, ja que es troben inactius.

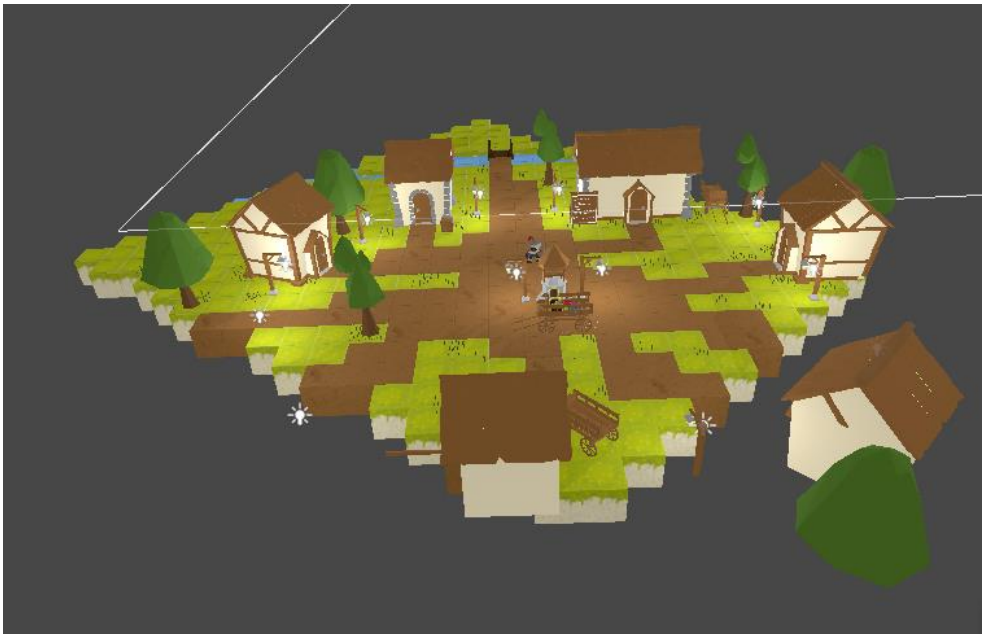


Figura 45: Captura del mapa de Kingdom Doom en temps d'execució

El moviment pel món del joc ha consistit en la creació d'una malla sobre el mapa que s'adapta segons els obstacles. En el moment que l'usuari fa un clic sobre qualsevol lloc de la malla, s'agafa la posició de la casella que ha clicat. Posteriorment s'avisa al personatge quina és la posició objectiu i calcula el millor camí fins aquesta posició.

Figura 46: Captura de la malla del mapa de Kingdom Doom

El camí creat a partir de la malla no té en compte les caselles sobre les quals es mouen els personatges. Per moure el personatge sobre aquestes caselles cal modificar el camí i crear-ne un de nou que faci passar el personatge per les caselles.

A continuació es mostra el pseudocodi on es discretitza el camí que pren el personatge en cel·les. El codi original es troba a l'Annex 11.1.

La següent funció recorre tots els punts que es creen automàticament sobre la malla que retornen el camí més curt possible per arribar al destí, i avancen cel·la a cel·la fins a la posició objectiu per crear un nou camí.

```
Funció modificaAgentPath():
	Mentre (camiCreat == false):
		Esperem un frame
	Fi Mentre

	Si (camiCreat == true):
		Per cada posició del camí:
			posicioInicial = posició[i]
			posicioFinal = posició[i+1]
			Mentre (posicioFinal != posicioInicial):
				Distancia = distanciaEnCeles(posicioInicial, posicioFinal)
				Si (Distancia.x == Distancia.y):
					posicioDiagonal  = AvançaEnDiagonal()
				Si (diagonalValida() == true):
					posicioInicial = posicioDiagonal
```

```
                        Altrament:
                              posicioCostat = AvançaDeCostat()
                        Si (costatValid () == true):
                              posicioInicial = posicioCostat
                        Altrament Si (diagonalValida()==false && costatValid()==false)
                              Retorna error
                  Fi Mentre
            Fi Per

      Retorna cami
Fi Funció
```

6.5 *Combats*

Els combats són gestionats per un script que defineix les posicions inicials dels personatges, l'ordre dels torns, avisa a cada personatge quan és el seu torn, calcula les caselles que són accessibles per cada personatge, etc.

6.5.1 *Moviment*

Per calcular el moviment dels personatges, donada la posició actual del personatge i els moviments que té disponibles, es calcularà casella a casella totes les direccions possibles fins a esgotar els moviments. Quan el camí que s'està explorant ja ha estat trobat amb un nombre de moviments igual o inferior a l'actual, no es calcula el camí de nou, ja que no es podrà trobar cap solució nova. El codi és molt simple, però té una velocitat de càlcul elevada.

A continuació es mostra el pseudocodi on es defineixen les cel·les on es pot desplaçar un personatge dins de combat. El codi original es troba a l'Annex 11.2.

És una funció recursiva que avança per les cel·les laterals, una darrera l'altre, fins a gastar tots els moviments, per crear tots els camins possibles que pot fer el personatge amb els moviments que l'hi queden.
Aquesta primera funció inicia la crida a la funció recursiva que calcula els camins possibles.

```
Funció buscaMovimentsPersonatge(posició, moviments):
      Retorna buscaCamins(cami, posició, 0, moviments)
Fi Funció
```

Llavors, la funció recursiva avança per les cel·les laterals, una darrera l'altre, fins a gastar tots els moviments, per crear tots els camins possibles que pot fer el personatge amb els moviments que l'hi queden.

```
Funció buscaCamins(camí, posició, movimentsActuals, movimentsMax):
      Si (posicioValida() == true):
               Camí.Afegir(posicio);

      movimentsActuals += 1
      Si (movimentsActuals <= movimentsMax):

            posicioDreta = avançaDreta()
            buscaCamins(camí, posicioDreta, movimentsActuals, movimentsMax)
            posicioEsquerra = avançaEsquerra()
            buscaCamins(camí, posicioEsquerra, movimentsActuals, movimentsMax)
            posicioAmunt = avançaAmunt ()
            buscaCamins(camí, posicioAmunt, movimentsActuals, movimentsMax)
            posicioAvall = avançaAvall ()
            buscaCamins(camí, posicioAvall, movimentsActuals, movimentsMax)


      Retorna camí
Fi Funció
```

6.5.2 *Habilitats*

El càlcul del rang de les habilitats és molt més complex que el dels moviments, ja que cal tenir en compte el tipus d'atac, el rang mínim i màxim i les caselles ocupades. Per calcular el rang, primer de tot una funció s'ocupa de crear una llista de totes les cel·les sense tenir en compte si estan ocupades o no. Una segona funció, a partir de la llista de les cel·les totals possibles, crea una segona llista de les cel·les que no són vàlides. Per últim una tercera funció és l'encarregada d'eliminar les cel·les invàlides de la llista de cel·les possibles, juntament amb totes les cel·les que es vegin afectades. En tot aquest procediment cal tenir en compte el tipus d'atac que es tracta, ja que hi ha atacs que travessen les parets o atacs que són únicament en línia.

Per saber quines cel·les es veuen afectades quan un atac no travessa parets i hi ha algun obstacle, s'han definit unes regles que cal seguir per poder eliminar totes les cel·les afectades de la llista. Quan un obstacle es troba al costat del personatge que utilitza l'habilitat, les cel·les crearan un con darrere l'obstacle, com més lluny està l'obstacle més tancat serà el con. A la Figura 47 es mostra el raonament utilitzat per aquest cas.

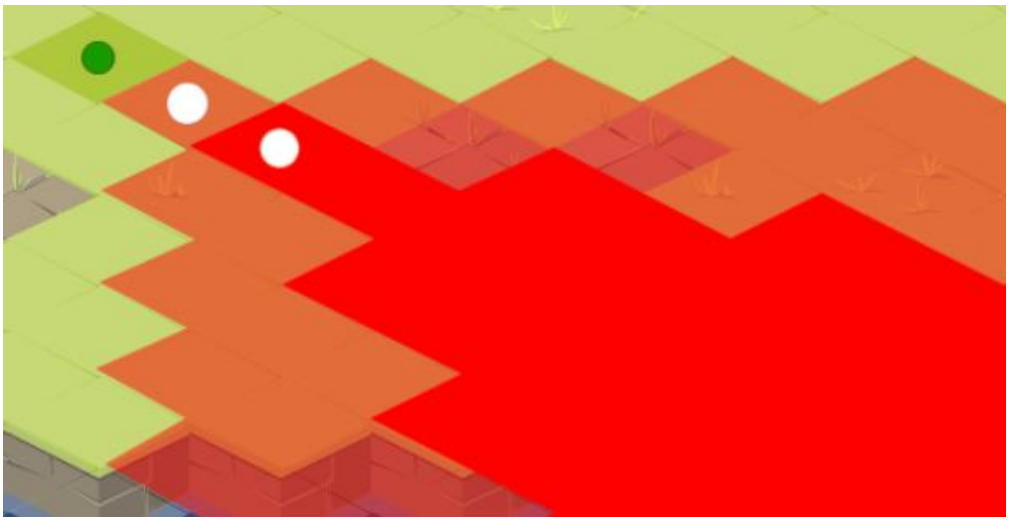


Figura 47: Captura del funcionament del càlcul del rang d'atacs (posició de costat)

A la imatge anterior se simula que el personatge de color verd vol llançar un atac, però té dos obstacles de color blanc al costat: les cel·les de color vermell clar són les que bloqueja el primer obstacle i les cel·les més fosques són les bloquejades pel segon obstacle. Quan l'obstacle es troba en una diagonal perfecta respecta del personatge que utilitza l'habilitat, el tractament de les cel·les afectades és molt semblant a l'anterior, ja que es crea un con darrere l'obstacle. Aquest cas és igual a la Figura 47, però els obstacles de color blanc es troben en diagonal amb el personatge de color verd.

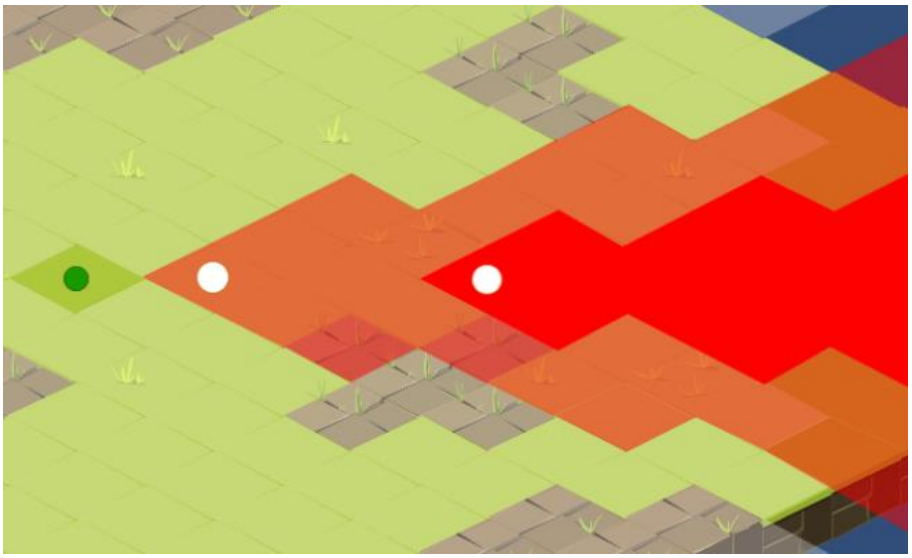


Figura 48: Captura del funcionament del càlcul del rang d'atacs (posició en diagonal)

Per últim l'objecte es pot trobar en una diagonal no perfecta. Aquest últim cas també crea un con darrere la cel·la ocupada, però combina els dos raonaments anteriors, ja que avança en diagonal i de costat alhora.

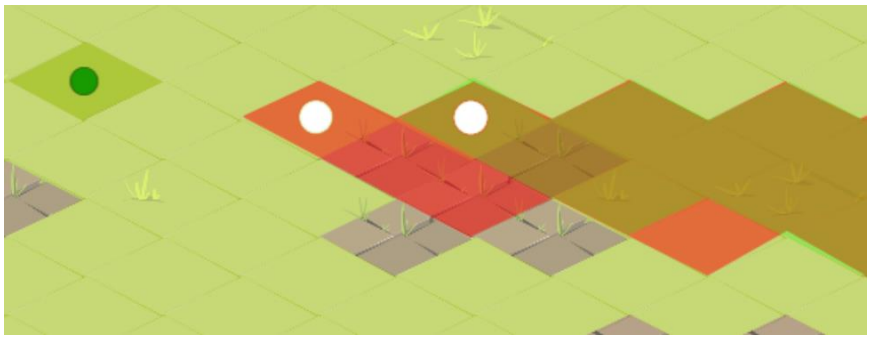


Figura 49: Captura del funcionament del càlcul del rang d'atacs (posició no concreta)

La imatge anterior mostra com el personatge de color verd, que vol llançar una habilitat, té dos obstacles que no estan ni en diagonal ni de costat. En aquest cas les cel·les vermell transparent mostren la línia de visió bloquejada pel primer obstacle i les cel·les de color verd transparent que es troben sobre les cel·les vermelles són les bloquejades pel segon obstacle. El càlcul de cel·les afectades per un obstacle està fet completament a mà. La idea principal era crear un obstacle en forma de con invisible que detectés quines cel·les estaven dins el con, però necessitava múltiples frames per col·locar l'obstacle i eliminar les cel·les. Així, finalment, es va crear una funció que calculava les cel·les afectades i eliminant-les de la llista.

Quan una habilitat té una àrea d'acció de més d'una cel·la, s'utilitzen les mateixes funcions per calcular quina és l'àrea que es veurà afectada per l'habilitat que utilitzarà el personatge.

A continuació es mostra el pseudocodi de les funcions que s'ocupen del càlcul del rang de les habilitats. El codi original es troba a l'Annex 11.3. La següent funció recull i modifica les cel·les que es troben dins el rang depenent del tipus d'atac o habilitat.

```
Funció BuscaRangAtacPersonatge(posició, rangMin, rangMax, tipus):
// posició = posició del personatge
      Si (tipus == 'N' OR tipus == 'I'):
            posicionsValides = retornaPosicionsRang(posició, tipus, rangMin, rangMax)
            posicionsInvalides = retornaPosicionsRangInvalides(posició,
                                                                posicionsValides, tipus)
            camí = eliminaPosicionsRangInvalides(posició, posicionsValides,
                                                 posicionsInvalides, tipus, rangMax)
      Altrament Si (tipus == 'L' OR tipus == 'X')
            Camí = retornaPosicionsRang(posició, tipus, rangMin, rangMax)
     Retorna Camí
Fi Funció
```

Depenent del tipus d'atac o habilitat, guarda les posicions que estan dins el rang. Les habilitats de tipus 'N' o 'I' són en àrea i les habilitats de tipus 'L' o 'X' són en línia recta.

```
Funció retornaPosicionsRang(posició, tipus, rangMin, rangMax):
       Si (tipus == 'N' OR tipus == 'I'):
             PosicioInicial1 = posició;
             PosicioInicial2 = posició;
             Per (i = 0; i <= rangMax; i++):
                   Per (j = 0; j <= (rangMax - i); j++):
                         PosicióDreta = avançarDreta(PosicioInicial1)
                         Camí.Afegir(posicioDreta)
                         PosicióEsquerra = avançarEsquerra(PosicioInicial1)
                         Camí.Afegir(posicioEsquerra)

                         PosicióDreta = avançarDreta(PosicioInicial2)
                         Camí.Afegir(posicioDreta)
                         PosicióEsquerra = avançarEsquerra(PosicioInicial2)
```

```
                        Camí.Afegir(posicioEsquerra)
                    Fi Per

                    PosicioInicial1 = avançarAmunt()
                    PosicioInicial2 = avançarAvall()
                Fi Per


        Altrament Si (tipus == 'L'):

            // direccions = [dreta, esquerra, amunt, avall]
            Per (i = 0; i < direccions.length; i++):

                posicioRang = posició + direcció
                j = 1;
                Mentre (esPosicioValida(posicioRang) AND j <= maxRang):
                    Si (j >= minRang AND j <= maxRang):
                        Cami.Afegir(posicioRang)

                    posicioRang = posició += direccio[i]
                Fi Mentre
            Fi Per

        Altrament Si (tipus == 'X'):
            // direccions = [dreta, esquerra, amunt, avall]
            Per (i = 0; i < direccions.length; i++):
                posicioRang = posició + direcció
                j = 1
                Mentre (j < maxRang):
                    Si (esPosicioValida(posicioRang) AND j>=minRang AND j<=maxRang):
                            Cami.Afegir(posicioRang)
                    posicioRang += direcció
                    j += 1
                Fi Mentre
            Fi Per
        Retorna Cami
Fi Funció
```

Comprova si les posicions són invàlides. En cas de ser-ho, les guarda i les retorna.

```
Funció retornaPosicionsInvalides(posició, posicionsValides, tipus):
	Si (tipus == ‘N’ OR tipus == ‘I’):
		Per cada posicioV de posicionsValides
			Si (esPosicioValida(posicioV) == false):

				posicionsInvalides.Afegeix(posicioV)
		Fi Per
	Retorna posicioV
Fi Funció
```

Les habilitats de tipus ‘N’ i ‘I’ requereixen eliminar les posicions invàlides. Les de tipus ‘N’ cal aplicar els càlculs que estan representats a les figures 47, 48 i 49, ja que no poden travessar les parets i cal calcular, per cada obstacle, totes les cel·les que es veuen afectades. En canvi les habilitats de tipus ‘I’ poden travessar les parets i només cal eliminar les cel·les on es trobin obstacles.

Quan un personatge que utilitza una habilitat de tipus ‘N’ i té un obstacle a un costat, avançarà des de la posició de l’obstacle en direcció oposada al personatge que ha llançat l’habilitat en línia recta eliminant totes les caselles que es troben per darrere l’obstacle. Un cop eliminades, es desplaçarà en lateral a la casella que toca i tornarà a eliminar les caselles que es trobin per darrere l’enemic. Això es repetirà una vegada i una altra fins a eliminar totes les caselles que es trobin a rang de l’habilitat.

```
Funció eliminaPosicionsRangInvalides (posició, posicionsValides, posicionsInvalides,
tipus, rangMax)
	Si (tipus == ‘N’):
		Per cada PosicioI de posicionsInvalides:

			Distancia = distanceCells(posició, posicioI)

			// Distancia:
				// x = -1 (Dreta)
				// x = 1 (Esquerra)
				// y = 1 (Abaix)
				// y = -1 (Adal)

			Si (distancia.x == 0 OR distancia.y == 0): // Es troba en un costat

				Diagonal1 = posicioI
				Diagonal2 = posicioI
```

```
Per (i = 0; i <= rangMax; i++):

        Diagonal1TMP = Diagonal1
        Diagonal2TMP = Diagonal2
        Per (j = 0; j <= rangMax - i; j++):
            Si (tagPos(diagonal1) == 'Mapa'):

                Cami.elimina(diagonal1TMP)
            Si (tagPos(diagonal2) == 'Mapa'):

                Camí.elimina(diagonal2TMP)
            Si (distancia.y == 0): // Diagonals en X

                Diagonal1TMP = avançaEnX()
                Diagonal2TMP = avançaEnX()
            Altrament: // Diagonals en Y

                Diagonal1TMP = avançaEnY()
                Diagonal2TMP = avançaEnY()
        Fi Per
        Per (w = 0; w < distancia; w++):
            Si (distancia.y == 0): // Diagonals en X

                Diagonal1 = avançaEnX()
                Diagonal2 = avançaEnX()
            Altrament: // Diagonals en Y

                Diagonal1 = avançaEnY()
                Diagonal2 = avançaEnY()
        Fi Per
        Si (distancia.y == 0): // Diagonals en X

            Diagonal1 = avançaEnY()
            Diagonal2 = atraçaEnY()
        Altrament: // Diagonals en Y

            Diagonal1 = avançaEnX()
            Diagonal2 = atraçaEnX()
Fi Per

    ...
```

Quan l'obstacle es troba en una diagonal, es calcula prèviament la distància diagonal, que definirà els desplaçaments laterals que utilitzarà posteriorment. Per eliminar les cel·les, comença desplaçant lateralment les vegades que s'han definit a l'inici i eliminant totes les cel·les. En acabar, des de la posició que havia començat el desplaçament lateral, es mourà en diagonal en la posició oposada al personatge que ha llançat l'habilitat.

```
                    ...
        Altrament Si (distancia.x == distancia.y): // Es troba en
        diagonal
              distanciaDiagonal = distancia.x + distancia.y
              Si (distanciaDiagonal >= 4):
                    bloquejarDespLateral = 2
              Altrament:
                    bloquejarDespLateral = 1
              diagonal = posicioI
              desplaçamentsLaterals = 0
              desplaçamentsDiagonals = 0
              Mentre (distanciaDiagonal+desplaçamentsLaterals<=rangMax):
                    Camí.elimina(diagonal)
                    Per (i = 0; i <= desplaçamentsLaterals; i++):
                          Camí.elimina(desplaçaX())
                          Camí.elimina(desplaçaY())
                    Fi Per
                    Diagonal = desplaçaXY()
                    desplaçamentsDiagonals++
                    Si  (desplaçamentsDiagonals>=bloquejarDespLateral)
                          desplaçamentsDiagonals = 0
                    Altrament
                          desplaçamentsLaterals += 1
              Fi Mentre
                    ...
```

Les diagonals no perfectes, es desplaça per l'eix 'X' o 'Y' depenent de la distància de cada un eliminant les cel·les, posteriorment, fa un desplaçament en l'eix contrari, sempre tenint en compte l'anterior, i torna a començar. S'eliminen les cel·les que es troben a la distància de l'obstacle fins al rang màxim de l'atac.

```
                                    ...
            Altrament // Es troba en diagonal no perfecte
                        distanciaActual = Max(distancia.x, distancia.y)
                        posicioLateral = posicioI
                        desplaçamentsVertials = 0
                        i = 0
                        Mentre (distanciaActual <= rangMax):
                              Camí.elimina(posicioLateral)
                              Per (j = 1; j <= desplaçamentsVertials; j++)
                                    posicioTMP
                                    Si (distancia.x > distancia.y):
                                          posicioTMP = avançaEnX(posicioLateral)
                                    Altrament
                                          posicioTMP = avançaEnY(posicioLateral)
                                    Camí.elimina(posicioTMP)
                              Fi Per
                              Si (i>=distanciaLateral):// Avançem en l'eix menor
                                    Si (distancia.x > distancia.y):
                                          posicioTMP = avançaEnY(posicioLateral)
                                    Altrament:
                                          posicioTMP = avançaEnX(posicioLateral)
                                    Camí.elimina(posicioTMP)
                                    desplaçamentsVerticals -= 1
                                    i = 0
                              Altrament: // Avançem en l'eix major
                                    Si (distancia.x > distancia.y):
                                          posicioTMP = avançaEnX(posicioLateral)
                                    Altrament:
                                          posicioTMP = avançaEnY(posicioLateral)
                                    Camí.elimina(posicioTMP)
                                    desplaçamentsVerticals += 1
                                    i += 1
                              distanciaActual += 1
                        Fi Mentre
      Fi Per
                                    ...
```

Les habilitats de tipus 'I' únicament eliminen les posicions invàlides, ja que travessen les parets.

```
                                        ...

      Altrament Si  (tipus == 'I')
            Per cada PosicioI de posicionsInvalides:
                  Camí.elimina(posicioI)
            Fi Per

      Retorna Cami
Fi Funció
                                        ...
```

6.6 *Personatges*

Tots els personatges es mouen lliurement pel món gràcies a la malla que el cobreix i permet trobar ràpidament els camins òptims. En els combats, el jugador i els enemics es comporten de manera diferent, ja que els enemics es controlen tots sols amb intel·ligència artificial. El personatge del jugador només pren una acció si el jugador ho decideix, i no fa res per si sol.

A continuació es mostraran els aspectes més essencials dels personatges, tant del jugador com dels enemics.

6.6.1 *Controls Combats*

- Player

  Els controls dels combats del personatge principal depenen completament del jugador. El personatge es manté a l'espera de rebre una instrucció del jugador. En el moment que rep aquesta instrucció, la du a terme i es torna a posar en espera. Aquest és el funcionament del personatge en combat, és molt simple, únicament espera les instruccions del jugador.

- Enemics

  Els enemics es controlen per la intel·ligència artificial. Tenen unes pautes que sempre segueixen depenent del tipus d'enemic. Si un enemic és

agressiu, seguirà les següents pautes en ordre: si pot atacar, ataca. En cas contrari, es desplaçarà el mínim possible fins a atacar a l'usuari. Si no existeix cap posició on pugui atacar, gastarà tots els passos en la direcció més propera al jugador. En cas que l'enemic sigui passiu, les pautes a seguir seran les següents: si pot atacar, ataca. En cas contrari comprova si existeix una posició en la qual pugui atacar, si és així es desplaçarà fins aquella posició i atacarà. Quan ja no pugui atacar, buscarà la posició més allunyada del jugador.

A continuació es mostra el pseudocodi de la intel·ligència artificial dels enemics. El codi original es troba a l'Annex 11.4.

L'enemic sempre espera uns segons entre accions, el torn el desenvolupa de pressa així que no cal fer totes les accions de cop. Un cop ha de decidir l'acció que ha de fer, primer de tot, prova d'atacar. Si no pot, però té moviments, depenent del tipus d'enemic i del mana que tingui, s'acostarà al jugador o bé s'allunyarà d'ell.

```
Funció desenvolupaTorn()
    Mentre (actiu == true)
      Si (enMoviment == false):
          tempsTranscurregut += Time.deltaTime;
          Si (tempsTranscurregut >= tempsEntreAccions):
            Si (haPogutAtacar() == false):
                Si (atributs.moviments > 0):
                  Si (tipus==‘Agresiu’ OR (tipus==‘Passiu’ AND quedaManaPerAtacar()==true))
                        Moved = moveCloseToPlayer()
                        Si (Moved == false)
                              finalitzaTorn()
                  Altrament Si (tipus==‘Passiu’ AND quedaManaPerAtacar()==false)
                        Moved = moveFarToPlayer()
                        Si (Moved == false)
                              finalitzaTorn()
                Altrament
                  finalitzaTorn()
            tempsTranscurregut = 0
    Fi Mentre
Fi Funció
```

6.6.2 *Animacions*

L'arbre d'animacions del jugador té un funcionament molt simple, l'animació de córrer s'activa únicament quan es desplaça, a través d'una variable que només pot ser verdadera (córrer) i falsa (parat). La resta d'animacions funcionen amb un disparador: en el moment que s'activa, es llença l'animació i torna a la posició inicial.

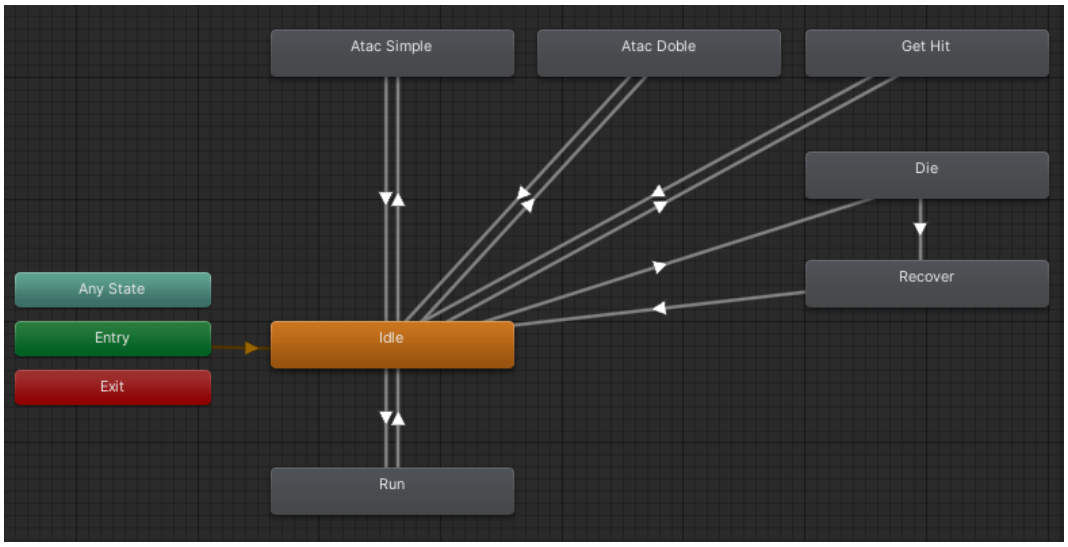


Figura 50: Captura arbre d'animacions del Player

Tots els enemics tenen un arbre igual, cada un apuntant a les seves animacions particulars. Totes les animacions tenen el mateix funcionament que les animacions del jugador, però l'animació que es llença en morir és diferent. Quan aquesta finalitza, crida una segona animació que no fa res més que cridar una funció especial que desactiva l'enemic. El motiu d'aquest funcionament és que alguns models 3D d'enemics que s'han utilitzat tenien les animacions bloquejades i per tant no es podien modificar, de tal manera que no es podia posar un disparador en finalitzar l'animació que crides la funció. La solució més simple era crear una animació que no afectes en res a l'enemic i que en el primer frame de l'animació desactives l'enemic, ja que estava mort.

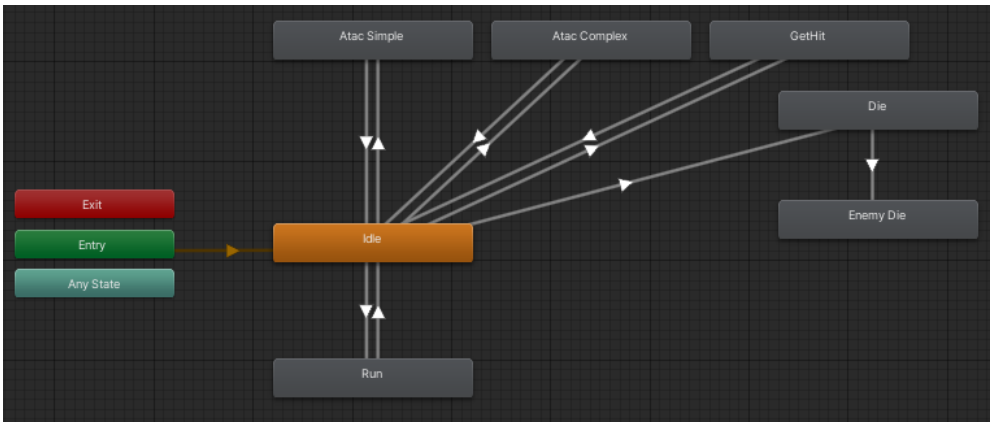


Figura 51: Captura arbre d'animacions dels enemics

6.6.3 *Grups d'enemics*

Els grups d'enemics o "mobs", estan repartits al llarg del mapa i són els encarregats de crear nous grups de monstres a través d'un script. En l'script del grup es defineix quins són els enemics i caps possibles d'invocar, que es defineixen amb un identificador numèric. També cal definir el nombre mínim i màxim de nivell dels enemics, que es genera aleatòriament per cada enemic. El mínim i el màxim nombre d'enemics a invocar també es defineix dins cada bloc, tots els caps que estiguin assignats s'invocaran. Finalment es defineixen les cel·les que es poden allunyar els enemics del centre del grup i el temps que es tardarà a invocar un nou grup de monstres quan el jugador hagi matat els invocats actualment.

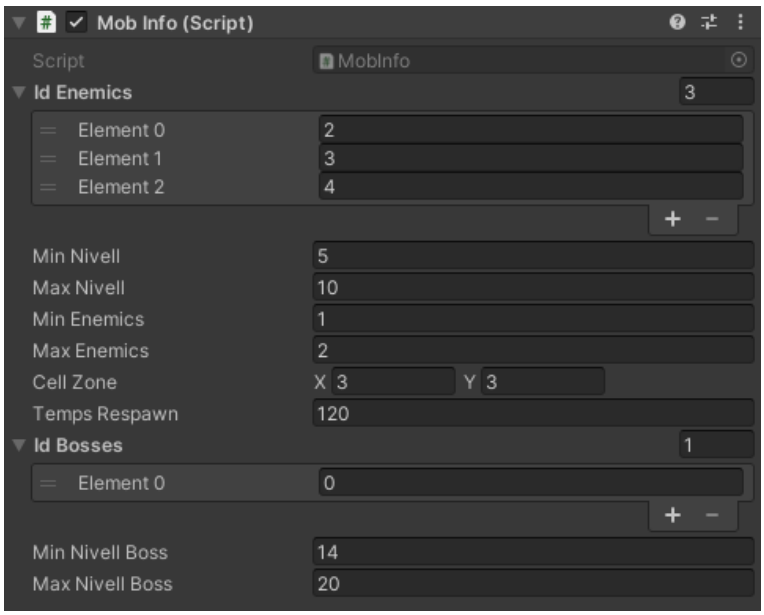


Figura 52: Captura del funcionament dels grups d'enemics

6.7 *Càmera*

La càmera és isomètrica i, per aconseguir aquesta vista, s'han creat dos objectes sense cap propietat, únicament donen la rotació a la càmera perquè la vista sigui isomètrica. L'estructura de la càmera és la que es pot veure a la Figura 53: són quatre objectes un dins l'altre, per aconseguir la vista isomètrica del joc.

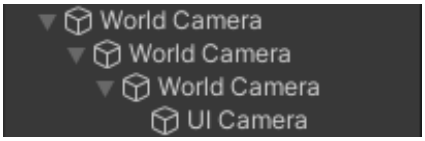


Figura 53: Estructura de la càmera

Per aconseguir la vista desitjada, el primer objecte està girat 270º en l'eix X. Aquesta rotació serveix perquè tots els objectes estan col·locats com si fos 2D. El segon objecte té una rotació de 30º en X i 315º en Y. Aquest segon objecte és el que gira la càmera de tal manera que la vista sigui isomètrica. El tercer objecte de l'estructura és la càmera que visualitza el món, és a dir, aquesta càmera no mostra cap de les interfícies d'usuari que apareixen dins el món del joc, per exemple: els textos de mana, vida i moviments dels combats o les informacions dels enemics. Per últim hi ha la càmera que únicament visualitza les interfícies d'usuari del món. D'aquesta manera, combinant les dues càmeres, aconseguim que les interfícies que es troben dins el món del joc sempre es visualitzin per davant de qualsevol objecte, si tot ho visualitzava la mateixa càmera, podien interferir objectes entre la visió del jugador i les interfícies.

6.8 *Missions*

Per agafar una nova missió el jugador s'ha d'apropar al taulell de missions, allà es publiquen les missions. Cada cert temps apareixerà una missió nova, fins a un màxim de 3 missions. El jugador pot decidir quina de les missions vol activar per ser completada i si en vol eliminar alguna per deixar pas a una nova missió. L'encarregat de controlar tot això és el taulell de missions amb un script on es guarden les dades de les missions i es creen quan cal. La missió activa s'encarrega el jugador d'actualitzar-la.

6.9 *Botiga i equipament*

En el poblat hi ha una botiga per intercanviar els diners que hagi acumulat el jugador. L'equipament que compri el podrà equipar en qualsevol moment des de la botiga. L'encarregat de controlar tot això és l'objecte botiga amb un script on té guardat els identificadors de tot l'equipament que s'hi pot comprar.

Abans d'iniciar el projecte es van definir uns objectius que calia assolir, en aquest apartat els valorarà si s'ha complert amb tots els objectius.

L'objectiu principal era desenvolupar les bases del joc, les parts més essencials que donessin una experiència més propera al que podria ser el joc final. El que calia desenvolupar per complir amb els objectius era clar, ara caldrà veure si s'ha complert amb tot el que es volia.

7.1 *Creació d'un mapa*

El joc en un principi era pensat per ser en 2D, però finalment es va decidir canviar al 3D, ja que facilitava moltes interaccions entre objectes. Indiferentment de si el joc era en 2D o 3D, el mapa havia de tenir un poblat, una zona que hi hagués varietat d'enemics i finalment una masmorra. El mapa que s'ha implementat conté tots aquests aspectes, un poblat central, un bosc que el rodeja i una masmorra que, en aquest, cas és una cova.

Figura 54: Mapa del Kingdom Doom

7.2 *Desplaçament pel mapa*

El moviment pel mapa era molt important, ja que havia de permetre moure lliurement a l'usuari i que sentis una alta llibertat de decisió en què vol fer i havia de tenir una fluïdesa tan alta com fos possible. El sistema de desplaçament pel mapa que s'ha implementat es fa a través d'una malla que cobreix tot el mapa, aquesta malla donat un punt inicial i un final calcula el trajecte més curt possible en un temps molt elevat. Aquesta eina era perfecta per l'objectiu que es volia. Ja que el joc es basa en un sistema de cel·les, s'ha aplicat una transformació al trajecte calculat per la malla que obliga el jugador a moure's per les cel·les. El resultat final

del moviment té una fluïdesa molt bona i és molt fàcil moure's mentre es manté el clic del ratolí i el jugador actualitza constantment el destí.

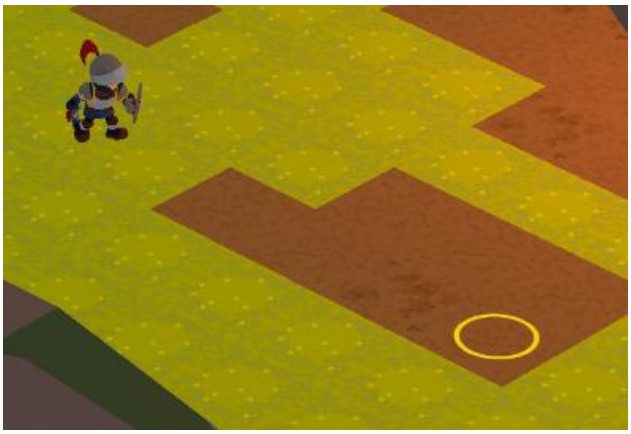


Figura 55: Captura del desplaçament pel mapa

7.3 *Sistema d'experiència i nivells*

Tenint en compte que el gènere del joc és JRPG, era necessari que hi hagués un sistema de progressió a través d'experiència i nivells i que aquests tinguessin un impacte sobre el joc. El sistema d'experiència i nivells implementat consisteix en el fet que l'usuari es veu recompensat amb experiència en guanyar un combat o finalitzar una missió. Quan té suficient experiència per avançar de nivell, automàticament s'intercanvia l'experiència que té el jugador per l'augment del nivell. Cada nivell atorga més vida i més poder en les habilitats al jugador, de tal manera que és molt important pujar de nivell per poder derrotar els enemics més i més poderosos que es trobarà pel camí.

Figura 56: Captura de l'animació en pujar de nivell

7.4 *Creació de grups d'enemics aleatoris*

La creació d'enemics havia de ser aleatòria al llarg del mapa, per tant calia que aleatòriament es creessin grups d'enemics als quals el jugador pugues derrotar. En el joc s'han definit diferents punts on apareixeran grups d'enemics totalment aleatoris. Aquests enemics es van movent pels voltants de la zona on han aparegut per donar vida al mapa.

Figura 57: Captura d'un grup d'enemics

7.5 *Combats en el món del joc*

Era essencial que el jugador pugues entrar en combat quan s'acostés als enemics. Aquestes lluites podien transportar al jugador a una instància del joc i no lluitar en el món, o bé, delimitar una zona dins el món i lluitar en el mateix mapa. Quan el jugador s'acosta molt a un enemic, entra en combat amb el conjunt d'enemics amb el que es troba. Les lluites es troben en el mapa del joc i no en una instància, ja que és més atractiu pel jugador, i cada terreny de joc pels combats serà completament diferent.

Figura 58: Captura de l'animació d'inici de combat

7.6 *Moviments dels combats*

En els combats era indispensable que el jugador i els enemics es poguessin moure lliurement pel terreny de joc, però amb un límit de moviments per torn. A l'usuari calia que se li mostressin totes les opcions possibles i ell decidís quina de totes les opcions era la que preferia. El sistema de moviment dels combats consisteix, tal com estava planejat inicialment, en calcular prèviament totes les caselles on es pot moure el personatge i mostrar-les al jugador. Llavors el jugador selecciona la casella que es vol desplaçar i el personatge fa el camí fins a arribar-hi. Els enemics funcionen de la mateixa manera, però la intel·ligència artificial és la que decideix quina és la casella més adequada per moure's.

Per millorar l'experiència del jugador, es va decidir mostrar tot el camí que faria el jugador fins a arribar a la cel·la desitjada, de tal manera que si el jugador posa el ratolí sobre una casella, el camí fins a arribar-hi canvia de color per saber quines són les cel·les per les quals passarà el personatge.

Figura 59: Captura dels moviments dels combats

7.7 *Habilitats dels combats*

El plantejament sobre les habilitats dels combats era que calia crear diferents tipus de rangs d'habilitats, que poden ser en línia i en àrea, havien de consumir manà, fer mal o curar i si era possible, que l'àrea d'acció també tingues diferents tipus de rangs. La implementació final de les habilitats és més complexa del que s'havia plantejat inicialment. A continuació enumerarem de què consten les habilitats:

- Rang d’habilitat

  El rang de les habilitats indica l’abast, amb unes caselles mínimes i unes màximes. A més a més, el tipus de rang, ens dirà si l’abast de l’atac és en àrea o en línia.

Figura 60: Captura del rang de les habilitats

- Consum de mana

  Totes les habilitats consumeixen mana, el consum dependrà del tipus d’habilitat.

- Atac i cura

  Les habilitats poden danyar i curar a l’objectiu, tant l’atac com la cura no són incompatibles en una sola habilitat.

- Àrea d’acció

  A més a més d’un rang d’habilitat, existeix una àrea d’acció. Aquesta determina la zona en la qual afectarà l’atac, que pot afectar a la casella seleccionada, a una àrea o en una línia. Aquesta àrea, igual que el rang de l’atac, té una distància mínima i màxima.

Figura 61: Captura de l'àrea d'acció de les habilitats

- Habilitats especials

El que fa que les habilitats siguin més complexes són les habilitats especials. Aquestes són totalment compatibles entre elles i amb l'atac i la cura, és a dir, una sola habilitat pot danyar i curar a l'objectiu i, a més a més, aplicar-l'hi les habilitats especials. Aquestes habilitats especials permeten treure mana o treure passos a l'objectiu. També es pot empènyer diverses cel·les en direcció contrària al personatge que ha llançat l'habilitat. Les habilitats especials també poden aplicar un estat a l'objectiu. Aquestes habilitats únicament les tenen els caps actualment. El cap Boletus, en tirar l'habilitat que resta tot el mana i els moviments al jugador, se li aplica un estat negatiu que evita que torni a tirar l'habilitat el temps que l'hi duri l'estat. L'Orc té un atac molt potent, però necessita un estat específic per poder ser utilitzat. Aquest estat s'aplica amb una habilitat especial que l'Orc pot utilitzar cada dos torns.

Figura 62: Captura dels detalls d'una habilitat

7.8 *Varietat d'enemics i caps*

Es buscava una varietat alta d'enemics, però no calia que fossin molt complexos. Pocs enemics molt complets faria el joc molt repetitiu. En canvi molta varietat d'enemics, tot i ser més simples, ajuda que el jugador no senti que es repeteixen les lluites una vegada i una altra. Un conjunt d'enemics pot ser completament diferent de l'anterior gràcies a la gran varietat d'enemics.

El joc té cinc enemics i dos caps diferents, tots i cada un tenen habilitats diferents, alguns amb més d'una habilitat per utilitzar. És més de l'esperat en un inici, ja que gràcies a la bona organització i planificació, crear un nou enemic era molt senzill, l'única complicació era crear el model 3D i les animacions, però s'han utilitzat els assets de Unity per agilitzar aquesta part de la creació d'enemics.

Figura 63: Captura de tots els enemics i caps

7.9 *Menús*

Era necessari que el joc tingues un menú inicial, un menú de pausa i un menú final, no havien de tenir moltes opcions, simplement calia que fossin fàcils d'entendre i d'utilitzar.

L'apartat de menús s'ha creat concorde a la idea inicial.

Figura 65: Menús de Kingdom Doom

7.10 *Interfícies d'usuari*

Les interfícies d'usuari són les que es troben ancorades a la pantalla, externes al món del joc. Les interfícies persistents del joc contemplades des d'un inici eren les referents a la vida, experiència, nivell, mana i passos del jugador. Finalment ha estat afegida una petita interfície referent a les missions.

Figura 66: Interfícies d'usuari persistents

Quan el jugador entra en combat, es mostra una segona interfície d'usuari. A la part inferior de la pantalla es mostra un botó pel moviment, totes les habilitats possibles per activar i el botó amb el temporitzador per passar de torn. Si el jugador passa el ratolí per sobre d'una de les habilitats, es mostra una descripció de l'habilitat, per ajudar al jugador a conèixer les habilitats.

Figura 67: Interfícies d'usuari dels combats

7.11 *Interfícies i animacions de món i de combat*

En la planificació inicial no estaven estrictament definides, algunes venien implícites amb certs funcionaments del joc, però s'han vist ampliades per donar una imatge més professional al joc.

A continuació es mostraran i s'explicaran les interfícies de món juntament amb les animacions creades per millorar l'aspecte del joc:

- Informació dels grups d'enemics
- Informació dels enemics en lluita
- Valors de dany, cura i mana de les habilitats
- Animació de les habilitats
- Animació de les torxes de la cova
- Animació de l'aparició de les cel·les dels combats

7.12 *Altres*

Aquest apartat no estava contemplat inicialment, però el joc, tot i contenir moltes coses, semblava molt simple per al jugador i s'ha decidit augmentar el contingut.
Finalment s'han creat les monedes que va recol·lectant el jugador quan guanya combats, juntament amb les missions que l'hi donen monedes i experiència com a recompensa. Les missions són completament aleatòries i demanen al jugador que derroti certs enemics. També s'ha afegit una botiga on el jugador pot intercanviar les monedes per equipament nou. L'equipament dota d'habilitats noves al jugador i l'hi proporciona més atac i més defensa.

7.13 *Errors durant el desenvolupament*

Com en qualsevol projecte, s'han trobat errors i s'han solucionat al llarg del desenvolupament. Per evitar la majoria de problemes és important planificar quin serà el contingut final del joc i tot el que es va desenvolupant tingui en compte tot el que es farà en un futur. Per aquest motiu no hi ha hagut grans errors ni s'han hagut de fer grans canvis per culpa d'un mal plantejament.
Els majors errors que ha tingut el videojoc han estat culpa de treballar amb nombres decimals, ja que en treballar amb Unity els valors decimals tendeixen a variar

constantment i s'ha de fer un tractament dels valors força exhaustius per evitar errors.

Inicialment el joc estava plantejat per ser construït en 2D, però en el moment de construir el mapa era complicat simular la profunditat. Això va fer que es plantegés a passar el joc al 3D, ja que solucionava tots els problemes de cop. Aquest canvi, per sort, va comportar menys temps de l'esperat, ja que únicament es va fer el canvi visual, però totes les col·lisions es van mantenir en 2D, de tal manera que els canvis eren mínims.

La idea del joc va sorgir fa anys, en el moment que ens van començar a sorgir els pensaments crítics respecte als videojocs. Com més experimentat respecte un tema, més et qüestiones les coses que fan els altres i planteges com podria estar fet o com es podria millorar. Per aquest motiu va sorgir la idea de crear un joc del gènere JRPG tàctic, ja que planteja un repte força gran i era un gran projecte per posar a prova les habilitats i coneixements adquirits al llarg dels anys d'estudi.

En els primers projectes que es creen, no solen estar ben estructurats o planejats i quan s'allarguen en el temps i es van ampliant, el desenvolupament es va alentint. Però en aquest projecte això ja es tenia en compte gràcies a l'experiència prèvia d'altres projectes desenvolupats. Tot i tenir un bon plantejament inicial i una bona organització, calia donar-hi més importància de la que ha tingut en un inici. Per la magnitud del projecte el plantejament ha sigut bo, però quan es desenvolupa un projecte un s'adona que segurament es podria haver tingut més control d'alguns aspectes des d'un inici.

El desenvolupament ha estat molt satisfactori, no només pel resultat final, sinó per l'estructura interna del joc. S'han aplicat moltes de les tècniques estudiades al llarg del curs, per la creació de classes, les funcions que fan els càlculs dels camins que ha de seguir el personatge, el rang de les habilitats dels personatges, etc.
El més important és l'experiència guanyada durant el temps que ha durat el projecte i seguir creant i augmentant els coneixements.

Per completar el joc, el més important seria desenvolupar la història, així que aquest seria el punt principal a desenvolupar en un futur.

L'equipament del jugador està parcialment implementat. En el treball futur caldria implementar les diferents parts de l'armadura i que tinguessin un impacte clar en el joc.

Finalment per acabar de completar el joc caldria afegir un inventari on es guardessin la multitud de recursos que podria recollir el jugador pel món. A més a més, els enemics donarien diferents recursos com a recompenses, d'aquesta manera es podrien crear missions no només de derrotar enemics sinó també d'entregar recursos. L'equipament caldria crear-lo amb els recursos, tot i que algun es podria comprar amb els diners.

[1] What is a Role-Playing Game (RPG)? – Definition
- 2018
- Realitzat per Techopedia: https://www.techopedia.com/definition/27052/role-playing-game-rpg

[2] What Are Role-Playing Games Even? How are they that?
- Sam Liberty
- 2017
- https://sa-liberty.medium.com/what-are-role-playing-games-even-how-are-they-that-50071c5552e2

[3] 15 Of The Best Turn-Based RPG's Of All Time
- 2019
- Realitzat per Unleash Gamer: https://unleashthegamer.com/best-turn-based-rpgs-of-all-time/#gref

[4] 12 recent games that prove turn-based RPGs aren't dead
- Thomas Sean
- 2020
- https://rpgoverload.com/turn-based-rpgs-arent-dead/

[5] Dofus
- 2004
- Realitzat per Ankama: https://www.dofus.com/es

[6] Wakfu
- 2012
- Realitzat per Ankama: https://www.wakfu.com/es/mmorpg

[7] Final Fantasy Tactics
- 2007
- Realitzat per Square Enix: http://dlgames.square-enix.com/fft/en/

[8] Dragon Quest Tact
- 2020
- Realitzat per Square Enix: https://dragonquest.square-enix-games.com/tact/en-gb/

[9] Dragon Quest VIII

- 2006
- Realitzat per Square Enix: https://www.jp.square-enix.com/dqsp/dq8/en/

11.1 *Funció discretització del camí del jugador*

```
    private IEnumerator modificaAgentPath()
    {
        int stopper = 20;
        int iStopper = 0;
        esperantCanviarTarget = true;
        while (!potCanviarTarget || agent.path.corners.Length < 2 || iStoppe
r < stopper)
        {
            iStopper++;
            yield return 0;
        }
        esperantCanviarTarget = false;

        if (agent.path.corners.Length >= 2)
        {
            List<Vector3> cornersList = new List<Vector3>(agent.path.corners
);
            cornersList.Add(agent.destination);
            Vector3[] corners = cornersList.ToArray();
            agent.ResetPath();

            // per cada parella de corners corner[x], corner[x+1] desplaçar-
se de cel·la en cel·la en direcció al corner[x+1]
            //  cal intentar agafar les cel·les que toquin la "linia" entre
corners
            //   per saber si toca la "linia", podem mirar la distancia de l
a cel·la a la linia
            List<Vector3> celes = new List<Vector3>();
            bool errorDiagonal = false; // Si en more en diagonal no podem
            bool errorCostat = false;
            for (int i = 0; i < corners.Length-2; i++)
            {
                Vector3 corner1 = WorldManager.Instance.roundPos( WorldManag
er.Instance.positionToCellSpace(corners[i], transform.tag) ); // Agafem el c
orner[x] i el discretitzem
                if (i == 0) celes.Add(corner1); // Afegir corner1 al inici

                Vector3 corner2 = WorldManager.Instance.roundPos( WorldManag
er.Instance.positionToCellSpace(corners[i+1], transform.tag) ); // Agafem el
 seguent corner i el discretitzem

                // Viatjar de corner1 a corner2 afegint noves cel·les
                Vector3 cornerDesp = corner1;
```

```
                bool errorCami = false;
                while (Vector3.Distance(cornerDesp, corner2) > .8f && !error
Cami)
                {
                    Vector2 direccio = new Vector2();
                    Vector2 distancia = WorldManager.Instance.distanceCells(
cornerDesp, corner2); // La distancia està en cel·les

                    if (distancia.x >= 0) direccio.x = -cellSpacing.x;
                    else direccio.x = cellSpacing.x;
                    if (distancia.y >= 0) direccio.y = -cellSpacing.y;
                    else direccio.y = cellSpacing.y;

                    // sempre que es pot avançar en diagonal es fa en diagon
al
                    // en cas que no es pugui, provar costats
                    Vector2 cornerTmp = cornerDesp;
                    if (!errorDiagonal && (Mathf.Abs(distancia.x) == Mathf.A
bs(distancia.y) || errorCostat)) // DIAGONAL
                    {
                        // Diagonal = 1,1; -1,1; 1,-1; -1,-1
                        cornerTmp.x += direccio.x;
                        cornerTmp.y += direccio.y;
                        if (!WorldManager.Instance.esPosicioValidaPersonatge
(cornerTmp))
                        {
                            Vector2 cornerXTmp = cornerTmp;
                            cornerXTmp.x += direccio.x;
                            cornerXTmp.y -= direccio.y;

                            Vector2 cornerYTmp = cornerTmp;
                            cornerYTmp.x -= direccio.x;
                            cornerYTmp.y += direccio.y;

                            if (WorldManager.Instance.esPosicioValidaPersona
tge(cornerXTmp) && WorldManager.Instance.esPosicioValidaPersonatge(cornerYTm
p))
                            {
                                Vector2 distanciaXTmp = WorldManager.Instanc
e.distanceCells(cornerXTmp, corner2);
                                int distanciaX = (int) Mathf.Abs(distanciaXT
mp.x) + (int) Mathf.Abs(distanciaXTmp.y);
                                Vector2 distanciaYTmp = WorldManager.Instanc
e.distanceCells(cornerXTmp, corner2);
                                int distanciaY = (int) Mathf.Abs(distanciaYT
mp.x) + (int) Mathf.Abs(distanciaYTmp.y);

                                if (distanciaX < distanciaY) cornerTmp = cor
nerXTmp;
                                else cornerTmp = cornerYTmp;
```

```
                            }
                            else if (WorldManager.Instance.esPosicioValidaPe
rsonatge(cornerXTmp))
                            {
                                cornerTmp = cornerXTmp;
                            }
                            else if (WorldManager.Instance.esPosicioValidaPe
rsonatge(cornerYTmp))
                            {
                                cornerTmp = cornerYTmp;
                            }
                        }

                        if (WorldManager.Instance.esPosicioValidaPersonatge(
cornerTmp) && !celes.Contains(WorldManager.Instance.roundPos(cornerTmp)))
                        {
                            cornerDesp = WorldManager.Instance.roundPos(corn
erTmp);
                            celes.Add(cornerDesp);
                            errorCostat = false;
                            errorDiagonal = false;
                        }
                        else
                        {
                            errorDiagonal = true;
                        }
                    }
                    else // COSTAT
                    {
                        // Costat = 1,0; -1,0; 0,1; 0,-1

                        if (Mathf.Abs(distancia.x) > Mathf.Abs(distancia.y))
 // Ens movem en X
                        {
                            Vector2 cornerXTmp = cornerTmp;
                            cornerXTmp.x += direccio.x;
                            if (!WorldManager.Instance.esPosicioValidaPerson
atge(cornerXTmp)) // Si X no és correcte, provem amb Y
                            {
                                cornerXTmp = cornerTmp;
                                cornerXTmp.y += direccio.y;
                                if (!WorldManager.Instance.esPosicioValidaPe
rsonatge(cornerXTmp))
                                {
                                    cornerXTmp = cornerTmp;
                                    cornerXTmp.y -= direccio.y;
                                }
                            }

                            cornerTmp = cornerXTmp;
```

```
                        }
                        else // Ens movem en Y
                        {
                            Vector2 cornerYTmp = cornerTmp;
                            cornerYTmp.y += direccio.y;
                            if (!WorldManager.Instance.esPosicioValidaPerson
atge(cornerYTmp)) // Si Y no és correcte, provem amb X
                            {
                                cornerYTmp = cornerTmp;
                                cornerYTmp.x += direccio.x;
                                if (!WorldManager.Instance.esPosicioValidaPe
rsonatge(cornerYTmp))
                                {
                                    cornerYTmp = cornerTmp;
                                    cornerYTmp.x -= direccio.x;
                                }
                            }

                            cornerTmp = cornerYTmp;
                        }

                        if (WorldManager.Instance.esPosicioValidaPersonatge(
cornerTmp) && !celes.Contains(WorldManager.Instance.roundPos(cornerTmp)))
                        {
                            cornerDesp = WorldManager.Instance.roundPos(corn
erTmp);

                            celes.Add(cornerDesp);
                            errorCostat = false;
                            errorDiagonal = false;
                        }
                        else
                        {
                            errorCostat = true;
                        }
                    }

                    if (errorDiagonal && errorCostat)
                    {
                        celes = new List<Vector3>();
                        errorCami = true;;
                    }
                }

                // Afegir corner2 al final
                if (!celes.Contains(corner2)) celes.Add(corner2);
            }
            if (celes.Count > 0) celes.Remove(celes[0]);

            varMoveFromTo = MoveFromTo(celes);
            StartCoroutine( varMoveFromTo );
```

```
        }
        else agent.ResetPath();
    }
```

```
    public Dictionary<Vector2, List<Vector2>> buscaMovimentsPersonatge(Vecto
    r2 posicio, int moviments)
    {
        movimentsValids = new Dictionary<Vector2, List<Vector2>>(); // reinic
iem viariable
        comprovaPosicionsMoviment(new List<Vector2>(), posicio, 0, moviments)
; // retornem totes les posicions possibles
        return movimentsValids;
    }

    private void comprovaPosicionsMoviment(List<Vector2> posicioPath, Vector
2 posicio, int movimentsActuals, int maxMoviments)
    {
        posicio = WorldManager.Instance.roundPos(posicio); // Arrodonim a 2 d
ecimals

        if (esPosicioValida(posicio) || movimentsActuals == 0) // Si casella
correcte o és la primera
        {
            List<Vector2> posicioPath_tmp = new List<Vector2>(posicioPath);
            bool continuar = false;

            if (movimentsActuals == 0) continuar = true;
            else if (!movimentsValids.ContainsKey(posicio)) // Afegim casella
 (la primera no s'agafa)
            {
                posicioPath_tmp.Add(posicio);
                movimentsValids.Add(posicio, posicioPath_tmp);
                continuar = true;
            }
            else if (posicioPath_tmp.Count+1 < movimentsValids[posicio].Count
)
            {
                posicioPath_tmp.Add(posicio);
                movimentsValids[posicio] = new List<Vector2>(posicioPath_tmp)
;
                continuar = true;
            }

            if (continuar)
            {
                movimentsActuals++;
                if (movimentsActuals <= maxMoviments)
                {
                    Vector2 posTopRight = new Vector2(posicio.x - cellSpacing
.x, posicio.y);
                    comprovaPosicionsMoviment(posicioPath_tmp, posTopRight, m
ovimentsActuals, maxMoviments);
```

```
                    Vector2 posBotRight = new Vector2(posicio.x + cellSpacing
.x, posicio.y);
                    comprovaPosicionsMoviment(posicioPath_tmp, posBotRight, m
ovimentsActuals, maxMoviments);
                    Vector2 posBotLeft = new Vector2(posicio.x, posicio.y - c
ellSpacing.y);
                    comprovaPosicionsMoviment(posicioPath_tmp, posBotLeft, mo
vimentsActuals, maxMoviments);
                    Vector2 posTopLeft = new Vector2(posicio.x, posicio.y + c
ellSpacing.y);
                    comprovaPosicionsMoviment(posicioPath_tmp, posTopLeft, mo
vimentsActuals, maxMoviments);
                }
            }
        }
    }
```

```
    public List<Vector2> buscaRangAtacPersonatge(Vector2 posicio, int rangMin
    , int rangMax, char tipus)
    {
        List<Vector2> posicionsFinals = new List<Vector2>();
        // segons el tipus fer una cosa o altra
        if (tipus == 'N' || tipus == 'I')
        {
            List<Vector2> posicionsValides = retornaPosicionsRang(posicio, ti
pus, rangMin, rangMax);
            List<Vector2> posicionsInvalides = retornaPosicionsRangInvalides(
posicio, posicionsValides, tipus);
            posicionsFinals = eliminaPosicionsRangInvalides(posicio, posicion
sValides, posicionsInvalides, tipus, rangMax);
            posicionsFinals = posicionsValides;
        }
        else if (tipus == 'L' || tipus == 'X')
        {
            posicionsFinals = retornaPosicionsRang(posicio, tipus, rangMin, r
angMax);
        }

        return posicionsFinals;
    }



    private List<Vector2> retornaPosicionsRang(Vector2 posicio, char tipus, i
    nt minRang, int maxRang)
    {
        List<Vector2> posicionsRang = new List<Vector2>();

        if (tipus == 'N' || tipus == 'I')
        {
            Vector2 posInicial1 = posicio;
            Vector2 posInicial2 = posicio;
            for (int i = 0; i <= maxRang; i++)
            {
                Vector2 posRang1_1 = posInicial1;
                Vector2 posRang1_2 = posInicial1;

                Vector2 posRang2_1 = posInicial2;
                Vector2 posRang2_2 = posInicial2;

                for (int j = 0; j <= (maxRang - i); j++)
                {
                    if ((i+j) >= minRang && (i+j) <= maxRang) posicionsRang.A
dd(posRang1_1);
```

```
                    posRang1_1 = new Vector2(posRang1_1.x + cellSpacing.x, po
sRang1_1.y); // Right Down

                    if (j > 0)
                    {
                        if ((i+j) >= minRang && (i+j) <= maxRang) posicionsRa
ng.Add(posRang1_2);
                    }
                    posRang1_2 = new Vector2(posRang1_2.x - cellSpacing.x, po
sRang1_2.y); // Left Up

                    if (i > 0)
                    {
                        if ((i+j) >= minRang && (i+j) <= maxRang) posicionsRa
ng.Add(posRang2_1);
                        posRang2_1 = new Vector2(posRang2_1.x + cellSpacing.x
, posRang2_1.y); // Right Down

                        if (j > 0)
                        {
                            if ((i+j) >= minRang && (i+j) <= maxRang) posicio
nsRang.Add(posRang2_2);
                        }
                        posRang2_2 = new Vector2(posRang2_2.x - cellSpacing.x
, posRang2_2.y); // Left Up
                    }
                }

                posInicial1 = new Vector2(posInicial1.x, posInicial1.y + cell
Spacing.y); // Right Up
                posInicial2 = new Vector2(posInicial2.x, posInicial2.y - cell
Spacing.y); // Left Down
            }
        }
        else if (tipus == 'L')
        {
            //Vector2[] posicions = {new Vector2(1,0),new Vector2(-
1,0),new Vector2(0,1),new Vector2(0,-1)};
            Vector2[] posicions = {new Vector2(cellSpacing.x,0),new Vector2(-
cellSpacing.x,0),new Vector2(0,cellSpacing.y),new Vector2(0,-cellSpacing.y)};

            for (int i = 0; i < posicions.Length; i++)
            {
                Vector2 posRang = posició + posicions[i];
                int j = 1;
                while (esPosicioValida(posRang) && j <= maxRang)
                {
                    if (j >= minRang && j <= maxRang) posicionsRang.Add(posRa
ng);
                    posRang += posicions[i];
```

```
                    j++;
                }

                if (!esPosicioValida(posRang) && getTagPos(posRang) != "Mapa"
 && j <= maxRang) posicionsRang.Add(posRang);
            }
        }
        else if (tipus == 'X')
        {
            //Vector2[] posicions = {new Vector2(.5f,.25f),new Vector2(-
.5f,.25f),new Vector2(-.5f,-.25f),new Vector2(.5f,-.25f)};
            Vector2[] posicions = {new Vector2(cellSpacing.x,0),new Vector2(-
cellSpacing.x,0),new Vector2(0,cellSpacing.y),new Vector2(0,-cellSpacing.y)};

            for (int i = 0; i < posicions.Length; i++)
            {
                Vector2 posRang = posicio + posicions[i];
                int j = 1;
                while (j < maxRang)
                {
                    if (esPosicioValida(posRang) || (!esPosicioValida(posRang
) && getTagPos(posRang) != "Mapa"))
                    {
                        if (j >= minRang && j <= maxRang) posicionsRang.Add(p
osRang);
                    }
                    posRang += posicions[i];
                    j++;
                }
            }
        }

        if (minRang == 0)
        {
            if (!posicionsRang.Contains(posicio)) posicionsRang.Add(posicio);
        }
        else posicionsRang.Remove(posicio);

        return posicionsRang;
    }

    private List<Vector2> retornaPosicionsRangInvalides(Vector2 posicio, List
    <Vector2> posicionsValides, char tipus)
    {
        List<Vector2> posicionsInvalides = new List<Vector2>();
        if (tipus == 'N' || tipus == 'I')
        {
            foreach (Vector2 posValida in posicionsValides)
            {
                if (!esPosicioValida(posValida))
```

```
                {
                    if (posValida != posicio) posicionsInvalides.Add(posValid
a);
                }

            }
        }

        return posicionsInvalides;
    }


    private List<Vector2> eliminaPosicionsRangInvalides(Vector2 posicio, List
    <Vector2> posicionsValides, List<Vector2> posicionsInvalides, char tipus,
     int maxRang)
    {
        List<Vector2> posicionsFinals = posicionsValides;

        if (tipus == 'N')
        {
            foreach (Vector2 posInvalida in posicionsInvalides)
            {
                Vector2 distancia = WorldManager.Instance.distanceCells(posic
io, posInvalida);
                // Distancia:
                // x = -1 (Dreta)
                // x = 1 (Esquerra)
                // y = 1 (Abaix)
                // y = -1 (Adal)

                if (distancia.x == 0 || distancia.y == 0) // Es troba a un co
stat (amunt, aball, esquerra o dreta), en linia recta
                {
                    // Eliminar en forma de T
                    // Si 1,1, la forma s'aplica cada casella
                    // Si 2,2, la forma s'aplica cada 2 caselles, etc.

                    int distanciaTmp = 0;
                    if (distancia.y == 0) distanciaTmp = (int) Mathf.Abs(dist
ancia.x);
                    else distanciaTmp = (int) Mathf.Abs(distancia.y);

                    float desplacamentX = cellSpacing.x;
                    if (distancia.x > 0) desplacamentX = -desplacamentX;
                    float desplacamentY = cellSpacing.y;
                    if (distancia.y > 0) desplacamentY = -desplacamentY;

                    Vector2 diagonal1 = posInvalida;
                    Vector2 diagonal2 = diagonal1;
```

```
                    for (int i = 0; i <= (maxRang - distanciaTmp); i++) // ca
lculem les diagonals que hem de fer
                    {
                        // diagonals
                        Vector2 tmpDiagonal1 = diagonal1;
                        Vector2 tmpDiagonal2 = diagonal2;
                        for (int j = 0; j <= (maxRang - distanciaTmp) - i; j+
+) // comprovem les linies de la diagonal
                        {
                            if (tmpDiagonal1 != posInvalida || ( tmpDiagonal1
 == posInvalida && getTagPos(tmpDiagonal1) == "Mapa")) posicionsFinals.Remove
(tmpDiagonal1);
                            if (tmpDiagonal2 != posInvalida || ( tmpDiagonal2
 == posInvalida && getTagPos(tmpDiagonal2) == "Mapa")) posicionsFinals.Remove
(tmpDiagonal2);

                            if (distancia.y == 0) // Diagonals en X
                            {
                                tmpDiagonal1 = new Vector2(tmpDiagonal1.x + d
esplacamentX, tmpDiagonal1.y);
                                tmpDiagonal2 = new Vector2(tmpDiagonal2.x + d
esplacamentX, tmpDiagonal2.y);
                            }
                            else // Diagonals en Y
                            {
                                tmpDiagonal1 = new Vector2(tmpDiagonal1.x, tm
pDiagonal1.y + desplacamentY);
                                tmpDiagonal2 = new Vector2(tmpDiagonal2.x, tm
pDiagonal2.y + desplacamentY);
                            }
                        }

                        for (int w = 0; w < distanciaTmp; w++) // avançem qua
nt calgui per fer la diagonal
                        {
                            if (distancia.y == 0) // Diagonals en X
                            {
                                diagonal1 = new Vector2(diagonal1.x + desplac
amentX, diagonal1.y);
                                diagonal2 = new Vector2(diagonal2.x + desplac
amentX, diagonal2.y);
                            }
                            else // Diagonals en Y
                            {
                                diagonal1 = new Vector2(diagonal1.x, diagonal
1.y + desplacamentY);
                                diagonal2 = new Vector2(diagonal2.x, diagonal
2.y + desplacamentY);
                            }
                        }
```

```
                        // avançem en diagonal
                        if (distancia.y == 0) // Diagonals en X
                        {
                            diagonal1 = new Vector2(diagonal1.x, diagonal1.y
+ desplacamentY);
                            diagonal2 = new Vector2(diagonal2.x, diagonal2.y
- desplacamentY);
                        }
                        else // Diagonals en Y
                        {
                            diagonal1 = new Vector2(diagonal1.x + desplacamen
tX, diagonal1.y);
                            diagonal2 = new Vector2(diagonal2.x - desplacamen
tX, diagonal2.y);
                        }
                        //diagonal1 = new Vector2(diagonal1.x + (desplacament
X * 2), diagonal1.y);
                        //diagonal2 = new Vector2(diagonal2.x, diagonal2.y +
(desplacamentY * 2));
                    }
                }
                else if (Mathf.Abs(distancia.x) == Mathf.Abs(distancia.y)) //
 Es troba en una diagonal
                {
                    // Avançem en diagonal, (per saber els passos que comprov
em: distancia en diagonal - desplaçaments laterals)

                    // Si distancia = 2 (1,1)
                    //  0, Diagonal 1 Costat, 2, 2, 3, 4, 4, 5, 6, 6, 7...
                    //  sumem 3 vegades, 1 no sumem, repetim

                    // Si distancia = 4 (2,2) (o més)
                    //  0, Diagonal 1 Costat, 1, 2, 2, 3, 3, 4, 4, 5, 5...
                    //  sumem 2 vegades, 1 no sumem, repetim

                    int distanciaDiagonal = (int) Mathf.Abs(distancia.x) + (i
nt) Mathf.Abs(distancia.y);
                    int bloquejarDespLaterals = 3;
                    if (distanciaDiagonal >= 4) bloquejarDespLaterals = 2;

                    // Distancia:
                    // x = -1 (Dreta)
                    // x = 1 (Esquerra)
                    // y = 1 (Abaix)
                    // y = -1 (Adal)
                    float desplacamentX = cellSpacing.x;
                    if (distancia.x > 0) desplacamentX = -desplacamentX;
                    float desplacamentY = cellSpacing.y;
                    if (distancia.y > 0) desplacamentY = -desplacamentY;
```

```
                    // Si diagonal = -X: -x, -y i -x, y
                    // Si diagonal = X: x, y i x, -y
                    // Si diagonal = -Y: -x, -y i -x, y
                    // Si diagonal = Y: x, y i x, -y
                    float desplacamentLateralX = cellSpacing.x;
                    if (distancia.x < 0) desplacamentLateralX = -
desplacamentLateralX;
                    float desplacamentLateralY = cellSpacing.y;
                    if (distancia.y < 0) desplacamentLateralY = -
desplacamentLateralY;

                    Vector2 diagonal = posInvalida;
                    int desplacamentsDiagonals = 0;
                    int desplacamentsLaterals = 0;
                    while ( (distanciaDiagonal - desplacamentsLaterals) <= ma
xRang)
                    {
                        if (diagonal != posInvalida || (diagonal == posInvali
da && getTagPos(diagonal) == "Mapa")) posicionsFinals.Remove(diagonal);

                        for (int i = 1; i <= desplacamentsLaterals; i++) // a
vançem de costat
                        {
                            Vector2 costat1 = new Vector2(diagonal.x + (despl
acamentLateralX * i), diagonal.y);
                            posicionsFinals.Remove(costat1);

                            Vector2 costat2 = new Vector2(diagonal.x, diagona
l.y + (desplacamentLateralY * i));
                            posicionsFinals.Remove(costat2);
                        }

                        // avançem en diagonal
                        diagonal = new Vector2(diagonal.x + desplacamentX, di
agonal.y + desplacamentY);
                        distanciaDiagonal += 1;

                        desplacamentsDiagonals++;
                        if (desplacamentsDiagonals >= bloquejarDespLaterals)
desplacamentsDiagonals = 0;
                        else desplacamentsLaterals++;
                    }
                }
                else
                {
                    int distX = (int) Mathf.Abs(distancia.x);
                    int distY = (int) Mathf.Abs(distancia.y);
                    int distanciaActual = Mathf.Max(distX, distY);
```

```
                    int distanciaLateral = Mathf.Max(distX, distY);
                    if (distanciaActual >= 5 && Mathf.Min(distX, distY) < (Ma
thf.Max(distX, distY) / 1.5)) // > 5 caselles amb el player i min < 2/3 de ma
x
                    {
                        distanciaLateral = Mathf.Max(distX, distY) - (Mathf.M
ax(distX, distY) - Mathf.Min(distX, distY));
                    }

                    // Desplaçament Lateral:
                    float desplacamentX = cellSpacing.x;
                    if (distancia.x > 0) desplacamentX = -desplacamentX;
                    float desplacamentY = cellSpacing.y;
                    if (distancia.y > 0) desplacamentY = -desplacamentY;

                    // Repetim de distanciaActual fins maxRang
                    int desplacamentsVerticals = 0;
                    int i = 0;
                    Vector2 posicioLateral = posInvalida;
                    while (distanciaActual <= maxRang)
                    {
                        if (posicioLateral != posInvalida || (posicioLateral
== posInvalida && getTagPos(posicioLateral) == "Mapa")) posicionsFinals.Remov
e(posicioLateral);

                        // Pas 1: Avançem de posActual per l'eix més gran (di
stanciaX > distancia Y) x vegades
                        if (i > 0)
                        {
                            for (int j = 1; j <= desplacamentsVerticals; j++)
                            {
                                Vector2 posicioVertical = new Vector2();
                                //if (distX > distY) posicioVertical = new Ve
ctor2(posicioLateral.x + (desplacamentX * j), posicioLateral.y - (desplacamen
tY * j));
                                //else posicioVertical = new Vector2(posicioL
ateral.x - (desplacamentX * j), posicioLateral.y + (desplacamentY * j));
                                if (distX > distY) posicioVertical = new Vect
or2(posicioLateral.x + (desplacamentX * j), posicioLateral.y);
                                else posicioVertical = new Vector2(posicioLat
eral.x, posicioLateral.y + (desplacamentY * j));
                                posicionsFinals.Remove(posicioVertical);
                            }
                        }

                        // Pas 2: Avançem sobre l'eix més gran o el menor
                        if (i >= distanciaLateral) // Avançem en l'eix contra
ri
                        {
```

```
                            //if (distX > distY) posicioLateral = new Vector2
(posicioLateral.x + desplacamentX, posicioLateral.y - desplacamentY);
                            //else posicioLateral = new Vector2(posicioLatera
l.x - desplacamentX, posicioLateral.y + desplacamentY);
                            if (distX > distY) posicioLateral = new Vector2(p
osicioLateral.x, posicioLateral.y + desplacamentY);
                            else posicioLateral = new Vector2(posicioLateral.
x + desplacamentX, posicioLateral.y);
                            desplacamentsVerticals--;
                            i = 0;
                        }
                        else // Avançem en l'eix més gran
                        {
                            //posicioLateral = new Vector2(posicioLateral.x +
 desplacamentX, posicioLateral.y + desplacamentY);
                            if (distX > distY) posicioLateral = new Vector2(p
osicioLateral.x + desplacamentX, posicioLateral.y);
                            else posicioLateral = new Vector2(posicioLateral.
x, posicioLateral.y + desplacamentY);
                            desplacamentsVerticals++;
                            i++;
                        }

                        distanciaActual++;
                    }
                }
            }
        }
        else if (tipus == 'I')
        {
            foreach (Vector2 posInvalida in posicionsInvalides)
            {
                if (getTagPos(posInvalida) == "Mapa") posicionsFinals.Remove(
posInvalida);
            }
        }

        return posicionsFinals;
    }
```

```
    private IEnumerator desenvolupaTorn()
    {
        while (actiu)
        {
            if (!enMoviment)
            {
                tempsAccioAnterior += Time.deltaTime;
                if (tempsAccioAnterior >= tempsEntreAccions)
                {
                    if (!haPogutAtacar())
                    {
                        if (atributs.getMovimentsActual() > 0) // Si queden m
oviments:
                        { // El moviment pot dependre del tipus enemic (passi
u, agresiu)
                            if (movimentsValids.Count < 1 && atributs.getMovi
mentsActual() > 0) buscaMovimentsPossibles();
                            List<Vector2> posicionsValides = new List<Vector2
>(movimentsValids.Keys);

                            if (tipus == 'A' || (tipus == 'P' && quedaManaPer
Atacar()))
                            {
                                if (!moveCloseToPlayer(posicionsValides))
                                {
                                    PartidaManager.Instance.finalitzarTornAct
ual();
                                }
                            }
                            else if (tipus == 'P' && !quedaManaPerAtacar())
                            {
                                if (!moveFarToPlayer(posicionsValides))
                                {
                                    PartidaManager.Instance.finalitzarTornAct
ual();
                                }
                            }
                        }
                        else if (atributs.getMovimentsActual() <= 0)
                        {
                            PartidaManager.Instance.finalitzarTornActual();
                        }
                    }

                    tempsAccioAnterior = 0;
                }
            }
```

```
            yield return 0;      // Leave the routine and return here in the
next frame
        }
    }
```

12.1 *Manual d'instal·lació*

A continuació s'explicaran els passos a seguir per instal·lar el joc:

1. Primer de tot, es necessitarà descarregar l'arxiu ZIP.
2. Un cop descarregat caldrà descomprimir aquest arxiu.
3. A la carpeta que s'haurà descomprimit hi trobarem el "Kingdom Doom.exe" que serà l'arxiu que caldrà executar.

12.2 *Manual d'usuari*

En iniciar el joc es mostrarà el menú principal, on permet iniciar una nova partida, continuar l'aventura i sortir del joc. Ja que és el primer cop que l'iniciem farem clic a iniciar partida.

Figura 68: Menú inicial de Kingdom Doom

Un cop iniciada la partida, per moure al jugador simplement farem clic a qualsevol lloc on no hi hagi un obstacle. D'aquesta manera es posarà a córrer automàticament fins a la posició que hem clicat.

Figura 69: Moviment del món de Kingdom Doom

Per iniciar alguna missió caldrà acostar-se al taulell de missions i fer clic al botó "E". S'obrirà un menú on ens dirà les missions que es poden activar o eliminar.

Figura 70: Taulell de missions de Kingdom Doom

Per entrar en un combat en dirigirem al bosc, tant per la sortida oest com la sortida nord del poblat. En apropar-nos a un grup d'enemics entrarem en combat automàticament.

Figura 71: Animació de lluita de Kingdom Doom

Un cop dins el combat, hi haurà un torn previ per col·locar a tots els personatges. Un cop hàgim visualitzat el terreny de combat, podrem fer clic a botó "START" per començar a combatre.

Figura 72: Botó START

Per moure al jugador per les cel·les caldrà fer clic al botó que trobem a la part inferior esquerra de la pantalla, posteriorment es mostraran les cel·les a les quals podem viatjar. Per moure al personatge caldrà fer clic a la cel·la desitjada.

Figura 73: Moviment dels combats de Kingdom Doom

Per derrotar als enemics caldrà utilitzar les habilitats, que es troben a la part inferior de la pantalla. En passar el ratolí per sobre una habilitat ens dirà tots els detalls. Per utilitzar una habilitat farem exactament el mateix que amb el moviment.

Figura 74: Habilitats dels combats de Kingdom Doom

Finalment podrem passar de torn abans de temps en fer clic al botó central inferior de la pantalla on hi ha el temps que queda per finalitzar el torn automàticament.

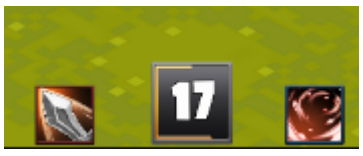


Figura 75: Botó finalitzar torn

Aquest és el funcionament principal del joc, amb aquestes quatre instruccions ja podrem iniciar la nostra aventura i acabar de descobrir tots els enemics, la botiga, l'equipament, etc. El funcionament de la botiga no s'ha explicat, ja que és molt semblant al de les missions i també és important que el jugador senti que hi ha coses noves per descobrir.